AI2-1

AI の具体的手法と開発基盤

－ 講義内容 －

- 講師 自己紹介
- 人工知能概論
- 人工知能の具体的手法
- 人工知能の開発プロジェクト
- 人工知能の開発基盤
- 演習：Python & Google Colaboratory の基礎

AIの具体的手法と開発基盤

2019年11月19日(火)
2019年11月20日(水)

新潟人工知能研究所 技術開発部

大目次

p03. 講座概要
p05. 講師紹介
p12. 人工知能概論
p18. 人工知能の具体的手法
p12. 人工知能の開発プロジェクト
p46. 人工知能の開発基盤

講座概要

株式会社新潟人工知能研究所
技術開発部

本講座の意義

- 現在のAIにおいて中核を担っている機械学習（マシン・ラーニング）や深層学習（ディープ・ラーニング）による分析手法（回帰、分類、クラスタリング、自然言語処理、画像認識）の概要を理解する。
- また、AIの実装を支える技術基盤について全体像を俯瞰するとともに、ハードウェア、ライブラリ（フレームワーク）、プラットフォーム、アプリケーションの各要素からAIの実装方法を理解する。

講師紹介

株式会社新潟人工知能研究所
技術開発部

株式会社新潟人工知能研究所

企業理念（ＮＡＩＬｉｓｍ）

『人と地域と地球に寄り添うAI』

ミッション：新潟にAI（人工知能）新会社設立。AI技術を駆使して、幅広くビッグデータを解析。データ・サイエンティストを育成して、新潟地域の生産性向上と地域創生に寄与していく

事業・代表紹介

- ビッグデータ解析
  - –教育系データ
- 人材育成
  - –インターン指導
  - –G検定対策講座
- プロダクト開発
  - –チャットボット,etc..

代表取締役 黒田達也

業務風景

2019年4月に事務所を新潟駅に直結するプラーカ２へと移転。アクセスもよく、すぐ下には大型書店もあって、研究開発には大変好都合です。

2019年10月に開催した「日本ディープラーニング協会主催G（ゼネラリスト）検定」の講座の様子。弊社は全社員がG検定の取得者です。

講師紹介

- 佐藤 修一
  - 株式会社新潟人工知能研究所
    - 技術開発部 部長
- 経歴
  - ㈱福島情報処理センター
    - システム開発部 エンジニア
    - インターネットプロバイダ部門
    - PCスクール部門 マネージャー
  - 新潟コンピュータ専門学校
    - 教務部 情報システム科 学科長
  - 2019年4月より現職

受講者の皆様へご協力のお願い

- より充実した講座にするために、一方向ではなく**受講者参加型**の要素を取り入れたく考えております
- 講義の途中でお近くの方と**アイデアを出し合ったり**、検討する時間を適宜、設けさせていただきます。
- これを機会に弊社、ならびに受講者の皆様と親睦を深め、**講義以上の物をお持ち帰り**いただけますと幸いです。よろしくお願いします。

ご提供の資料につきまして

- 資料には、一部、著作権の関係で印刷に含まれていないものもございますのでご容赦ください。
- ただし、ほとんどの資料は公式のテキストを参照するか、インターネットで検索していただくことで閲覧することができます。
- ぜひこれは！というキーワードはメモのご用意をお願いします

Memo

人工知能概論

株式会社新潟人工知能研究所
技術開発部

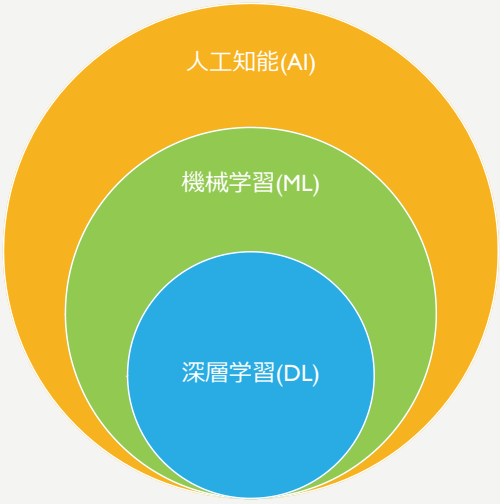
- この章ではｅラーニングの復習も兼ねて、特に今後の実習に必要な用語や知識を確認します

人工知能概論
～キーワード～

- 人工知能・機械＆深層学習
- 機械学習の変遷
- 深層学習の衝撃

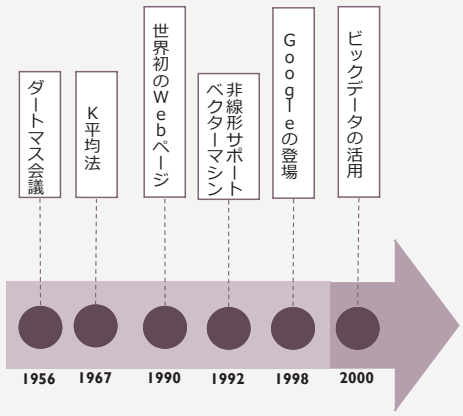
人工知能・機械学習・深層学習

- 関連キーワードの整理
  - 人工知能（Artificial Intelligence:AI）
    - 専門家の間でも明確な**定義はない**…
  - 機械学習（Machine Learning:ML）
    - 大量のデータから学習をすることで、**データに潜むパターンを発見**し、予測に活かすことができるプログラム
  - 深層学習（Deep Learning:DL）
    - 明確な定義はないが、一般的には機械学習に用いられる**ニューラルネットワークを多層**にしたもの（後述）
    - データの特徴を自動で捉える仕組み



機械学習（マシンラーニング）の変遷

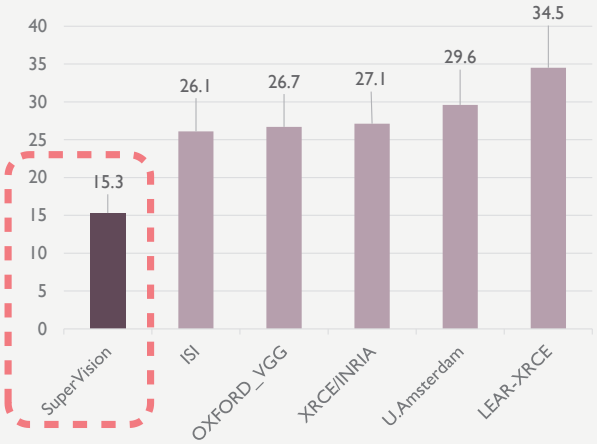
- 機械学習
  - 理論自体は長い研究の歴史がある
  - 近年、機械学習を大きく進化させるための**要因・環境が整った**のが背景
    - ビッグデータ、インターネット、IoT、GPU、センシング技術…
  - **レコメンデーション**や**迷惑メールフィルタ**も機械学習により実用化
  - さらに**深層学習（ディープラーニング）**が、ここから派生してブレイク



資料：「機械学習を取り巻く環境の変遷」

深層学習（ディープラーニング）の衝撃

- ILSVRC 2012
  - **画像認識の精度**を競うコンテスト
  - トロント大学のSuperVisionが**圧勝**
  - 当時は画像認識に機械学習を用いていたが、**特徴量（注目すべきデータの特徴）**を決めるのは人間
  - 特徴量を選ぶのは人間の職人芸
  - 世界中の研究者が**１%刻みの戦い**をしているなか**衝撃の結果**で優勝

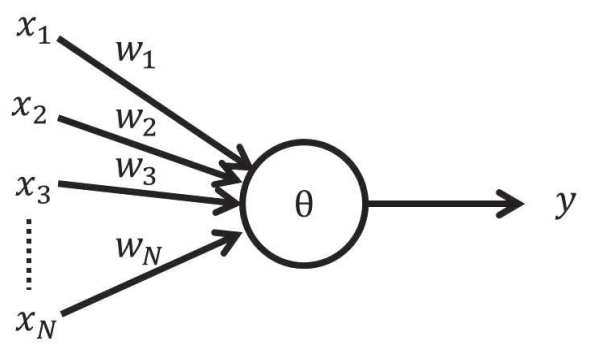


参考：DeepLearningってなんでこんなに流行ってるの？
- IT社員3人組によるリレーブログ
https://go-mount.hatenablog.com/entry/2018/08/24/235759

参考：翔泳社「深層学習教科書 ディープラーニング G検定（ジェネラリスト）公式テキスト」
p54 図2.29 「ILSVRC2012の結果」

ニューラルネットワーク

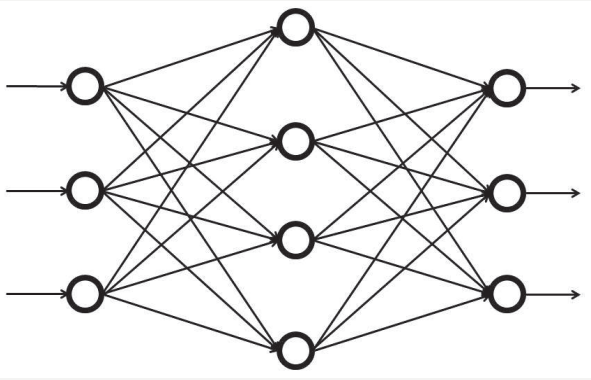
- 人間の脳の中の構造を模倣
  - ニューロン（神経細胞）
  - シナプス（接合部）
  - 発火（電気信号）
- 入力層／隠れ層／出力層
  - 単純パーセプトロン
  - 多層パーセプトロン
- 深層学習の登場以前は、そこまで人気がある手法ではなかった



出典：ニューラルネットワーク・DeepLearningなどの画像素材
http://nkdkccmbr.hateblo.jp/entry/2016/10/06/222245

深層学習（ディープラーニング）

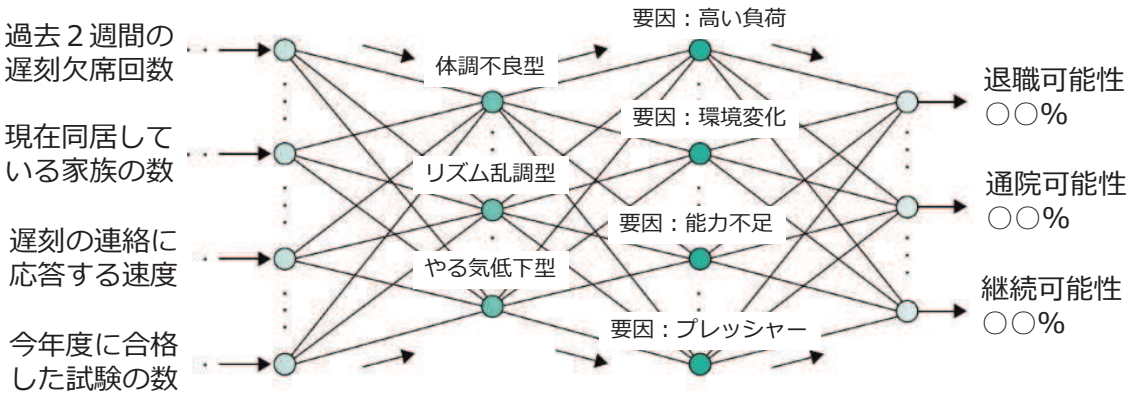
- 深層学習とは
  - 明確な定義は存在しないが、一般的には中間層が多層に重なり、層が深いニューラルネットワークを指すことが多い
  - ニューラルネットワークの元となるパーセプトロンは1958年に考案されていた。
  - ニューラルネットワークの多層化も早くから研究されており、1960年代に爆発的なブームを起こしたが、単純な問題しか解けないことが指摘されて下火になる。
  - しかしその後の改良と、ビッグデータ、ハードウェアの能力向上が追い風となり、第３次AIブームの火付け役として大躍進



出典：ニューラルネットワーク・DeepLearningなどの画像素材
http://nkdkccmbr.hateblo.jp/entry/2016/10/06/222245

深層学習による予測のイメージ

例：深層学習による「社員の退職予想」モデル
（注意．あくまでイメージで、実際の中間層は人間には理解困難なブラックボックス）



人工知能の具体的手法

株式会社新潟人工知能研究所
技術開発部

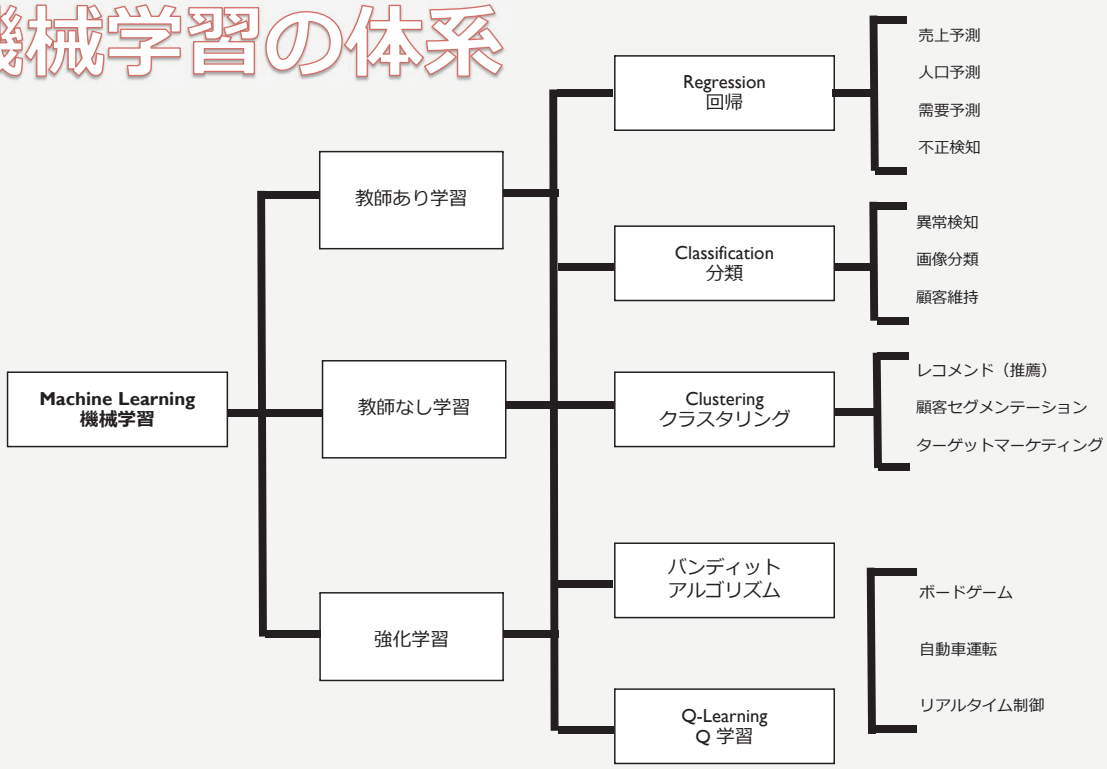
人工知能の具体的手法 ～キーワード～

- 教師あり学習／教師なし学習
- 学習とは
- 目的変数と説明変数（特徴量）
- 回帰／分類モデル
- クラスタリング
- 主成分分析／次元削減
- 強化学習
- ニューラルネットワーク
- 深層学習（ディープラーニング）
- 特徴表現学習

教師あり／なし学習・強化学習という分類方法

	教師あり学習	教師なし学習	強化学習
学習方法	正解（目的変数）がある訓練データを大量に分析して、パターンを予測するためのモデル（計算式）を導き出す	正解（目的変数）はなく、大量の訓練データの構造を分析し群＝クラスタに分けるモデル（計算式）を導き出す	人工知能の予測に対する誤差をフィードバックすることで、試行錯誤を繰り返しながら精度を自動で高めていく
学習内容	説明変数（入力）と目的変数（出力）の関係	データの構造 データの類似性	報酬（誤差が少ない場合に得られるスコア）を最大化する行動
具体的な手法	回帰モデル 分類モデル	クラスタリング 主成分分析／次元削減	Q学習 Deep Q-Network
用途の具体例	需要予測 解約予測 迷惑メールフィルタ 画像認識…	顧客特性分析 レコメンドエンジン データ分析前の処理…	株・FXなどのトレード 囲碁・将棋などのボードゲーム 自動運転や工作機械

機械学習の体系



機械学習のアルゴリズム

- 教師あり学習
  - ロジスティック回帰
  - 決定木
  - ランダムフォレスト
  - XGBoost
  - LightGBM
  - ニューラルネットワーク　…
- 教師なし学習
  - k-means、混合ガウス分布　…

名称	概要	予測の精度	データ必要量	速度	重要度可視化
ロジスティック回帰	事象の発生確率を予測	中	多	速	可能
決定木	1つの木構造で条件分岐	低	少	速	可能
ランダムフォレスト	複数の決定木による多数決	中	多	中	可能
XGBoost	ランダムフォレストの多数決に重み付け	高	多	中	可能
LightGBM	XGBoostを効率化	高	多	中	可能
ニューラルネットワーク	脳内を模した多層構造	高	中	遅	困難

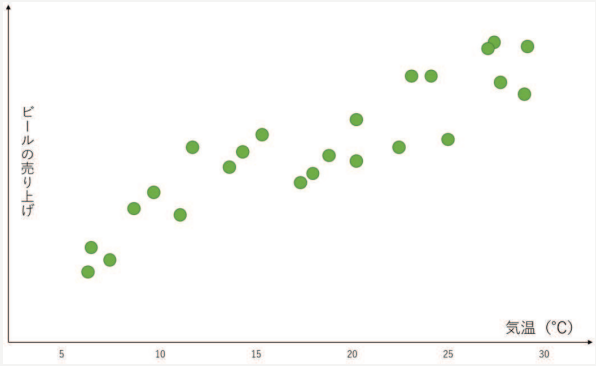
表：教師あり学習アルゴリズムの比較表

学習とは？

- 非 人工知能ソフトウェアの場合
  - プログラムに特定の計算式が組み込まれている→何度計算しても同じ結果になる（のが求められる）
- 人工知能ソフトウェアの場合
  - 既存データ（訓練データ）を大量に与えることにより、計算式（モデル）そのものが変化していく。
  - 訓練データが変化（増減）すれば、計算式（モデル）も変化していく。
- 訓練した計算式（モデル）を使用して、未知のデータに対して予測をさせる→この精度が重要
- これは訓練するたびに計算式が変化するので、訓練を行えば行うほどに予測の精度が上がる…
- という、うまい話はなく、場合によっては精度が下がることもあり得る（過学習という状況）

用語：目的変数と説明変数（特徴量）

- 目的変数
  - 予測したい対象のデータ
  - 例：ビールの売り上げ
- 説明変数
  - 目的変数を予測するために使うデータ群＝「特徴量」とも呼ぶ
  - 例：気温
- 特徴量を的確に設計できるか否かが予測の精度にかかってくる

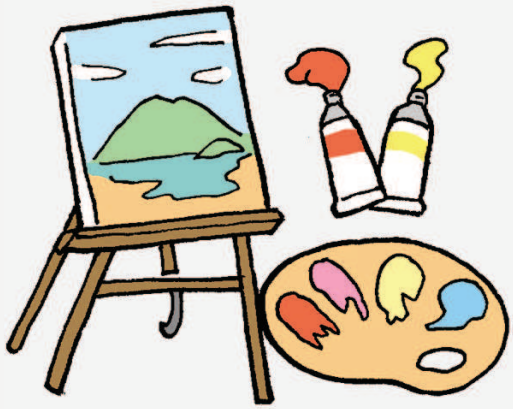


機械学習ができること

- 説明変数と目的変数の間にある関係性（パターン）を導き出し、モデル（計算式）を生成する
  - 人間には想像もできないような関係性を見つけることができれば…
- 人工知能の価値は「未知のデータに対して、モデル（計算式）を使って、どのくらい高精度の予測ができるか？」で決まる

機械学習が向かないこと

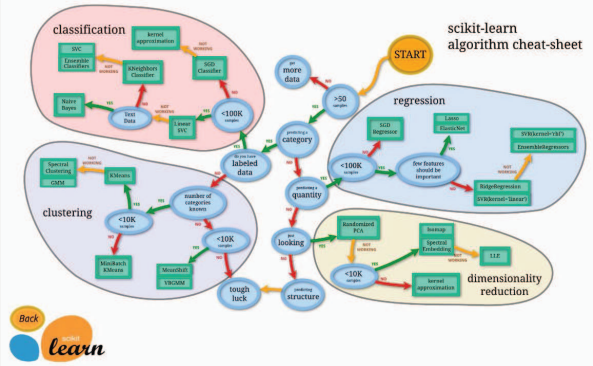
- 説明変数と目的変数が設定できない場合（そもそも関連性がない）
- 不測の事態ばかりが起こる事例
- ゼロから想像して何かを生み出す（過去に全く存在しないもの）
  - 有名画家の画風で絵を書く人工知能
- 人間の感情・倫理に起因する部分
- 人工知能にすることで逆に採算が取れなくなる事業。人間の方が…



参考： WIRED.jp「人工知能が描いた「レンブラントの新作」」
https://wired.jp/2016/04/14/new-rembrandt-painting/

機械学習が実現する４つの自動化

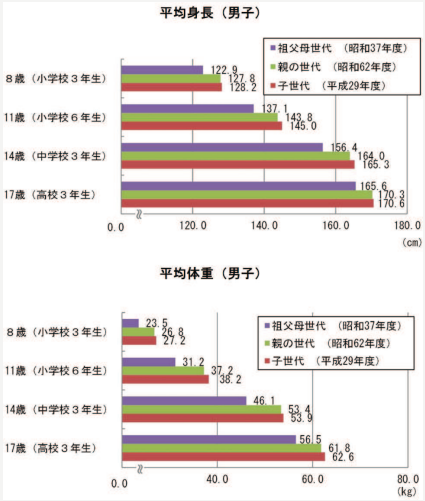
- 回帰（Regression）
  - 過去の実績値から未来の値を予測
- 分類（Classification）
  - あるカテゴリに属する確率を予測
- クラスタリング(Clustering)
  - 似ているデータをグループ化する
- 主成分分析／次元削減（Principal Component Analysis,PCA／Dimensionality Reduction）
  - 特徴量同士の関係性を把握する



出典：Choosing the right estimator — scikit-learn 0.21.3 documentation
https://scikit-learn.org/stable/tutorial/machine_learning_map/

回帰モデル

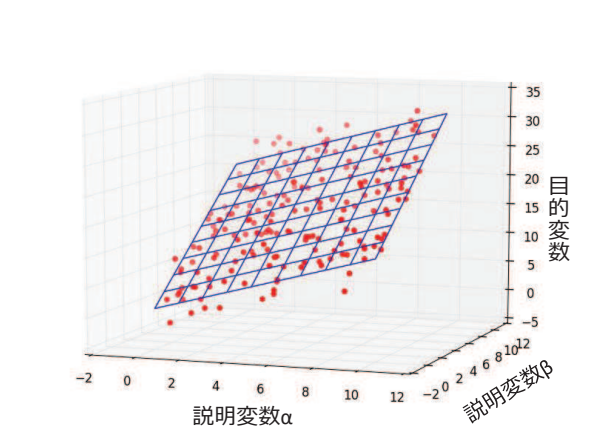
- 学習フェーズ
  - 訓練データ（例．身長と体重の組み合わせ）を大量に入力して、
  - 説明変数（身長）に対する
  - 目的変数（体重）の値を
  - 求めるモデル（計算式）を組立てる
- 予測フェーズ
  - 与えられた説明変数（身長）を
  - モデル（計算式）に当てはめて、
  - 未知の目的変数（体重）を予測する



出典：文部科学省「平成29年学校保健統計」
p3「発育状態の世代間比較（身長・体重）」
http://www.mext.go.jp/component/b_menu/other/__icsFiles/afieldfile/2018/03/26/1399281_01_1.pdf

回帰モデルの種類

- 単回帰分析
  - １つの説明変数から、１つの目的変数を導出する
    - 説明変数：身長
    - 目的変数：体重
- 重回帰分析
  - 複数の説明変数から、１つの目的変数を導出する
    - 説明変数：身長・年齢・体脂肪…
    - 目的変数：体重



回帰モデルの導入事例

- ビールメーカーの需要予測
  - アサヒビール × NEC
- 課題：販売機会を逃すことなく、かつ、無駄な在庫をなくしたい
- 従来：ベテラン社員の経験と勘
- 要件：需要予測を自動化したい
- 説明変数：過去の出荷量＆販売量、暦、天気、競合他社の動向…
- 目的変数：ビールの需要量
- 導入効果：誤差15%（ベテラン並）

参考：AI技術で実現する、精度の高い需要予測 | NEC
https://jpn.nec.com/profile/vision/story/03.html

分類モデル

- 学習フェーズ
  - 訓練データ（例．スポーツアスリートの身体能力と競技名）を大量に入力して、
  - 説明変数（身体能力＝身長・体重・短距離走のタイム・筋力…）に対する、
  - 目的変数（何のスポーツ選手か？）を
  - 求めるモデル（計算式）を組み立てる
- 予測フェーズ
  - 与えられた説明変数（身体能力）を
  - モデル（数式）に当てはめて、
  - 未知の目的変数（陸上選手の確率、野球の、アメフト選手の確率…）を予測する

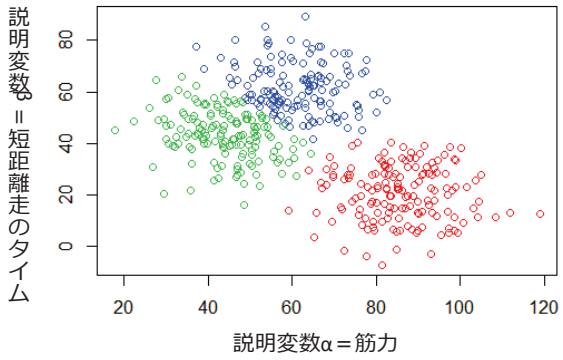
分類モデルの導入事例

- 携帯電話メーカーの解約予測
  - ドコモ関西「解約予兆モデル」（1998年）
- 説明変数：居住地、契約期間、通話時間、着信回数、コールセンターの利用回数…
- 目的変数：３ヶ月以内に解約する確率
- 導入効果：解約率を最大で１／３にした
- 発展：近年、目的変数が増加（料金プラン、利用サービス、固定電話の種別…）したことで「動画サービスに加入する確率」や「ドコモ光に乗り換える確率」など他サービスへの横展開が可能になった。
- 技術：当初は単純な予測モデルだったが、現在は機械学習で精度を向上させている

参考：ドコモのビッグデータ分析の歴史
「“宝の山”をマーケティングに活用」| ビジネスネットワーク.jp
https://businessnetwork.jp/Detail/tabid/65/artid/6315/Default.aspx

クラスタリング

- 学習フェーズ
  - 訓練データ（例．スポーツ選手の短距離走のタイム・筋力）を大量に入力して、
  - 説明変数（短距離走・筋力など）から、
  - 目的変数（選手が属するグループ＝クラスタ）を求めるモデル（計算式）を組む
- 予測フェーズ
  - 与えられた説明変数（短距離走・筋力）を、モデル（計算式）に当てはめて、
  - 未知の目的変数（選手が属するグループ＝クラスタ）がどれになるかを予測する



分類モデルとクラスタリングの違い

分類モデル

- あらかじめ決まったカテゴリのどれに属するかの確率を出す
- 訓練データは、
  - 説明変数：ある選手の身体能力
  - 目的変数：その選手のポジションのように、明確な正解が存在する（教師あり学習）未知のデータに対する予測の的中率が重要になる

クラスタリング

- どんなクラスタ＝群（まとまりの傾向）があるかを見つけ出す
- 訓練データには、
  - 説明変数：ある選手の身体能力
  - 目的変数：似た能力の選手らの群のように、明確な正解が存在しない（教師なし学習）その群れにどんな意味があるかは人間が考える

クラスタリングの導入事例

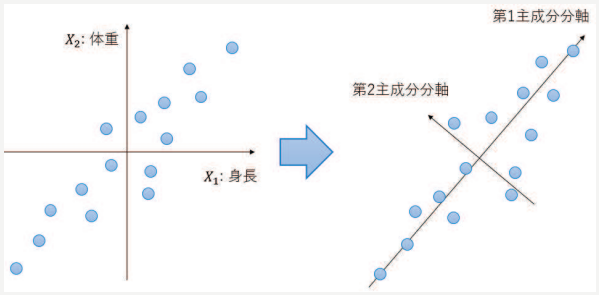
- スーツメーカーダイレクトメール最適化
  - はるやま商事 × SENSY（2016年）
- 説明変数：顧客の氏名、住所、連絡先、購入した商品の属性、時期…×３年分
- ポイント：仕立てを行うので、スーツ購入時に個人情報が確実に入手できる環境
- 目的変数：顧客特性ごとにクラスタリングし、好みに併せてDMをカスタマイズ
- 導入効果：20～30代向け「P.S.FA」店舗にて、通常DMと比べて売上27%、客数26%増。ただし、40～50代向けのはるやま通常店舗は、効果は限定的であった。
- 発展：DMの効果をフィードバック学習

写真はイメージです

参考：スーツブランド「P.S.FA」、人工知能「SENSY」を導入 ～DM最適化、ECレコメンドに活用：MarkeZine（マーケジン）
https://markezine.jp/article/detail/24582

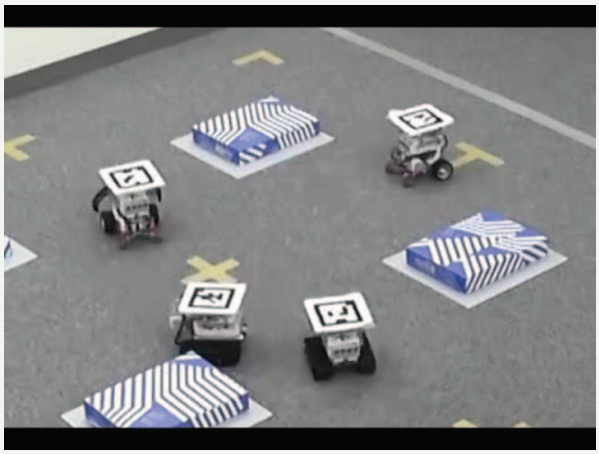
主成分分析／次元削減

- 主成分分析
  - 目的：データの大まかな傾向をつかむ
  - 図示：散布図上に「主成分軸」を引く
- 次元削減（次元圧縮）
  - 主成分分析をもとに、説明変数の次元（データが持つ項目の数）を削減する
  - 利用例：
    - 学生の模擬試験（国語・数学・社会・理科）の得点データ（4次元）を、2次元に削減した場合、理系と文系の成分になるかもしれない（ならないかもしれない）



強化学習

強化学習
人工知能の予測に対する誤差をフィードバックすることで、試行錯誤を繰り返しながら精度を自動で高めていく
報酬（誤差が少ない場合に得られるスコア）を最大化する行動
Q学習 Deep Q-Network
株・FXなどのトレード 囲碁・将棋などのボードゲーム 自動運転や工作機械
教師あり学習は訓練データ（教師データ）の準備に人間の手間がかかるため、近年は強化学習の研究も盛んである。



出典：Youtube： ScienceNews2016
「ディープラーニング　最先端の人工知能アルゴリズム」
https://www.youtube.com/watch?v=JGaHwOubY4Q

特徴量設計の難しさ

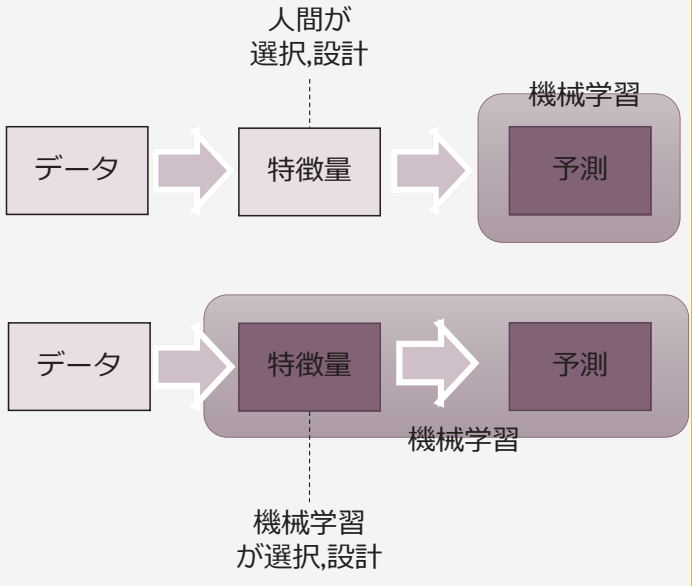
- 例）回帰の予測タスク
  - 「ビールの売り上げ」に対して「当日の気温」という特徴量は有り得そうだが…
  - 「ビールの売り上げ」に、「昨日の天気」は、果たして関係あるだろうか？
- 例）画像の認識タスク
  - 自動車が写っている写真からナンバープレートだけを取り出す…という画像認識
  - 「四角い白いプレート内」にいくつかの「数字」が記載されているという特徴？
  - でも白じゃないプレートもあるし、写真の写り方で「四角」に見えないことも…

画像認識における特徴表現学習

- ある写真から物体を認識させる
- 人間が特徴量を設計するには？
  - 物体のエッジ（端）を見つける
    - ピクセル単位で測定していき色合いの数値が急に変化するところ？
  - 物体の輪郭を捉える
    - エッジを繋ぎ合わせて、ある程度の「まとまり」となるところ？
  - 物体を認識させる
    - 輪郭を組み合わせて物体（人間や自動車、草花など）となる部分？

特徴表現学習

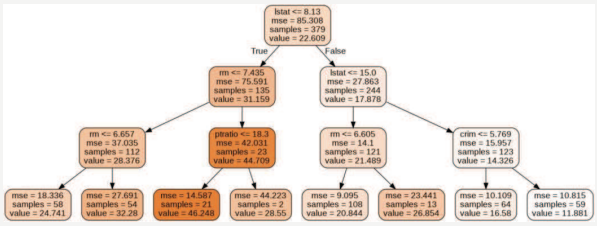
- 機械学習に与える特徴量を見つけ出すのが（職人芸で）難しい
  - では、特徴量そのものを機械学習自身に発見させたらどうだろう？
- 特徴表現学習というアプローチ
  - 深層学習は、「特徴表現学習」を行うアルゴリズムの一つ
  - データの特徴量を階層化し、それを組み合わせてモデルを構築する



深層学習の躍進と注意点

- 深層学習は確かに精度は高いが判断の根拠はブラックボックス
  - 「人工知能の判断根拠を明確にして欲しい」という要件（特に人命に関わる場合）に応えるのが困難
  - どうしても判断根拠が必要な場合は、多少の正解率を犠牲にしても、深層学習にこだわらず既存手法を採用するのがベターな場合もある
  - 補足キーワード「説明可能なAI」

参考：「AIの説明」の 現状とこれから（総務省）
http://www.soumu.go.jp/main_content/00058731l.pdf



参考：決定木による判断根拠の可視化

人工知能の開発プロジェクト

株式会社新潟人工知能研究所
技術開発部

- この章では、トップダウン的に人工知能を用いた開発プロジェクトの全体像を確認した後に、実際の実装工程におけるステップを解説します。
- また反面教師として、近年よく耳にする人工知能プロジェクトの失敗事例も確認しましょう。

人工知能の開発プロジェクト ～キーワード～

- 開発プロジェクトのフェーズ
- 人工知能プロジェクトの失敗
- 人工知能モデル構築ステップ
- 訓練に必要なデータを集める
- モデルの評価指標

人工知能開発プロジェクトのフェーズ

- ①構想（1～3ヶ月)
  - 課題（テーマ）の選定
  - パートナー企業の選定
  - 投資対効果（ROI）の検討
- ②PoC=概念検証（1～2ヶ月）
  - モックアップ（仮モデル）を構築
  - 利用可能なデータの検証
  - 予測精度の検証
- ③実装（3ヶ月～）
  - データ基盤の構築
  - 処理速度の検証
  - 精度の向上
  - ソリューション化
- ④運用（利用する限り継続）
  - 精度のモニタリング
  - モデルのチューニング
  - データの追加と再学習

人工知能×プロジェクト×失敗学

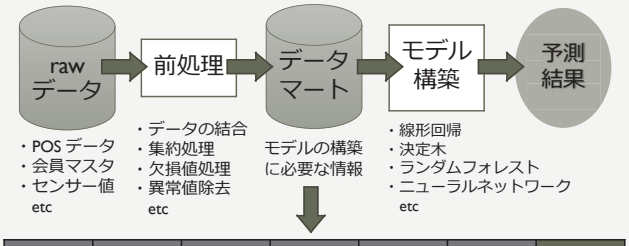
- 人工知能を用いてビジネスとして成功させるために必要な５つの“D”
- 頓挫するプロジェクトはこのどれかの観点が欠けている可能性あり
- 「AIプロジェクトの１／３は失敗する」（電通）
- 「AIプロジェクトの85%が失敗する理由」（Pactera Technologies）

- Define
  - どんなビジネス効果があるのか？
- Data
  - データの質・量は揃っているか？
- Develp
  - モデルの開発は現実的であるか？
- Deploy
  - 本番環境で実行に耐えうるのか？
- Drive
  - システムとして運用できるのか？

参考： LeapMind BLOG
「ディープラーニング/機械学習のビジネスフレームワーク「5D」」
https://leapmind.io/blog/2018/09/10/deeplearning5d/

人工知能モデル構築のステップ

- 前処理
  - GIGO（ガーベジイン・ガーベジアウト）＝ゴミを入れるとゴミが出てくる
  - ユーザ部門「データはあります」
    - 蓋を開けたらPDFデータだった…
    - 神エクセル（Excel方眼紙）だ…
  - 「データ分析の８割を占める」説
    - モデル（アルゴリズム）の良し悪しも大事だが前処理はもっと大事



顧客ID	年齢	性別 (男性:0, 女性:0)	来店回数	最終 来店後 経過日数	…	商品A 購入
1001	40	1	11	35	…	1
1002	29	0	5	12	…	0
1009			0		…	0
1010	18	0	12	10	…	1
…	…	…	…	…	…	…

説明変数
（パターンやルールを見つけるために使うデータ）

目的変数
（予測対象のデータ）

訓練に必要なデータを集める

- データ収集
  - 自社所有データ（1st party）
    - 分析が可能な形になっているか？
    - 一定数以上のデータ量があるか？
  - 他社保有データ（2nd party）
    - パートナー企業が収集したデータ
  - 政府・調査基幹データ（3rd party）
    - 有料の他、無料のオープンデータも
    - 分析用に成形されていない場合も…
  - 欲しいデータがない場合は、新たに取得する基盤を構築することも検討

出典：オープンデータ | 政府CIOポータル
https://cio.go.jp/policy-opendata

モデルの評価指標

- 回帰問題の精度を測る指標
  - RMSE（平均平方二乗誤差）
  - MAE（平均絶対誤差）
  - MSE（平均二乗誤差）
  - R2（決定係数）
  - RMSPE（平均平方二乗誤差率）
  - MAPE（平均絶対誤差率）
- 分類問題の精度を測る指標
  - 正解率（Accuracy）
  - 適合率（Precision）
  - 再現率（Recall）
  - F値（F-measure）

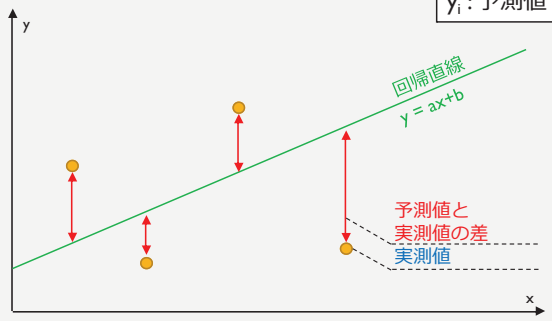
RMSE（二乗平均平方根誤差）

平均化された誤差の値（Root Mean Squared Error）
0に近いほど予測の誤差が小さく予測精度が高い

RMSEの式

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

y_i：実測値
$\hat{y}_i$：予測値



【例】株価の予測

日付	実測値	予測値	予測誤差
9月1日	21,000	21,100	-100
10月1日	22,000	21,800	-200
11月1日	23,000	23,300	+300

RMSE = 300
計算式：$\sqrt{(-100^2+-200^2+300^2) / 3} = \sqrt{90{,}000}$

R^2（決定係数）

予測値との実測値がどれほど一致しているかを表した指標
1に近いほど予測誤差が小さく予測精度が高い

R^2の式

$$R^2 = 1 - \frac{\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2}{\Sigma_{i=1}^{n}(y_i - \overline{y}_i)^2}$$

y_i：実測値
$\hat{y}_i$：予測値
$\overline{y}_i$：実測値の平均

【例】株価の予測

日付	実測値	予測値	予測誤差
9月1日	21,000	21,100	-100
10月1日	22,000	21,800	-200
11月1日	22,500	22,800	+300

R^2 : 0.955
計算式：$1 - (-100^2+-200^2+300^2) / (-1000^2+0\ \ +1000^2) = 1 - 90{,}000 / 2{,}000{,}000$

混合マトリックス

分類モデルの性能評価の目安であり、実際の結果と予測結果を集計した表

	予測は0 （Negative）	予測は1 （Positive）
実際は0 （Negative）	TN (True Negative)	FP (False Positive)
実際は1 （Positive）	FN (Fales Negative)	TP (True Positive)

TN（True Negative）　：実際は判別対象外のもの(０)を、0と正しく予測

FN（False Negative）　：実際は判別対象(１)のものを、0と誤って予測

FP（False Positive）　：実際は判別対象外のもの(０)を、1と誤って予測

TP（True Positive）　：実際は判別対象(１)のものを、1と正しく予測

正解率（Accuracy）

例：血圧・体温・血液検査の値から疾病を診断　※疾病(Positive) or 健康(Nagative)とする

正解率（Accuracy）＝全検査数の中で、どれだけ予測が当たったかの割合

	健康であると予測した （Negative）	疾病であると予測した （Positive）
実際は 健康であった （Negative）		
実際は 疾病であった （Positive）		

	健康であると予測した （Negative）	疾病であると予測した （Positive）
実際は 健康であった （Negative）	①　TN **1,000**	③　FP **20**
実際は 疾病であった （Positive）	②　FN **30**	④　TP **100**

Accuracy：0.957（計算式 = (①+④) / (①+②+③+④) = 1,100 / 1,150）

適合率（Precision）

例：血圧・体温・血液検査の値から疾病を診断　※疾病(Positive) or 健康(Nagative)とする

適合率（Precision）＝予測が正(Positive)の中で、実際に正(Positive)であったものの割合

	健康であると予測した（Negative）	疾病であると予測した（Positive）
実際は健康であった（Negative）		
実際は疾病であった（Positive）		

	健康であると予測した（Negative）	疾病であると予測した（Positive）
実際は健康であった（Negative）	① TN **1,000**	③ FP **20**
実際は疾病であった（Positive）	② FN **30**	④ TP **100**

Precision：0.833（計算式 = (④) / (③+④) = 100 / 120）

再現率（Recall）

例：血圧・体温・血液検査の値から疾病を診断　※疾病(Positive) or 健康(Nagative)とする

再現率（Recall）＝実際に正(Positive)の中で、正(Positive)と予測できたものの割合

	健康であると予測した（Negative）	疾病であると予測した（Positive）
実際は健康であった（Negative）		
実際は疾病であった（Positive）		

	健康であると予測した（Negative）	疾病であると予測した（Positive）
実際は健康であった（Negative）	① TN **1,000**	③ FP **20**
実際は疾病であった（Positive）	② FN **30**	④ TP **100**

Precision：0.769（計算式 = (④) / (②+④) = 100 / 130）

F値（F-measure）

例：血圧・体温・血液検査の値から疾病を診断　※疾病(Positive) or 健康(Nagative)とする

F値（F-measure）＝適合率と再現率の調和平均

	健康であると予測した（Negative）	疾病であると予測した（Positive）
実際は健康であった（Negative）		
実際は疾病であった（Positive）		

	健康であると予測した（Negative）	疾病であると予測した（Positive）
実際は健康であった（Negative）	① TN **1,000**	③ FP **20**
実際は疾病であった（Positive）	② FN **30**	④ TP **100**

Precision：0.8（計算式 = (2×適合率×再現率) / (適合率＋再現率）)
上記の計算式を展開すると ＝ 2×④ / (②＋③＋2×④) = 200 / 250

人工知能の開発基盤

株式会社新潟人工知能研究所
技術開発部

人工知能の開発基盤 ～キーワード～

・機械学習を利用するには

・AI開発基盤 俯瞰図2019

・2大機械学習プラットフォームの比較

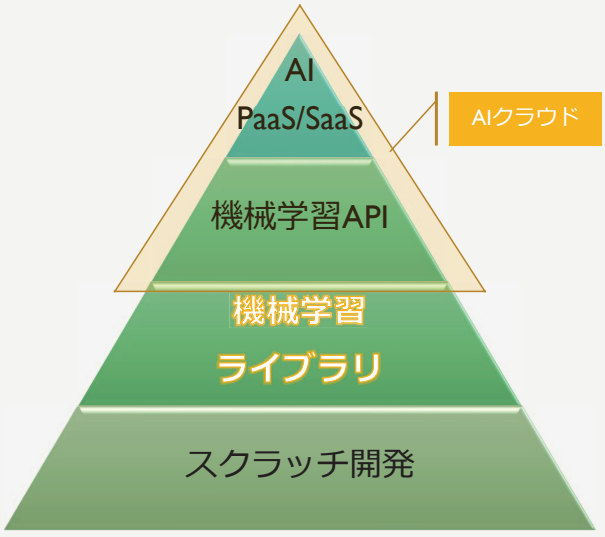
・使用する機械学習ライブラリ

・Python言語を利用するには

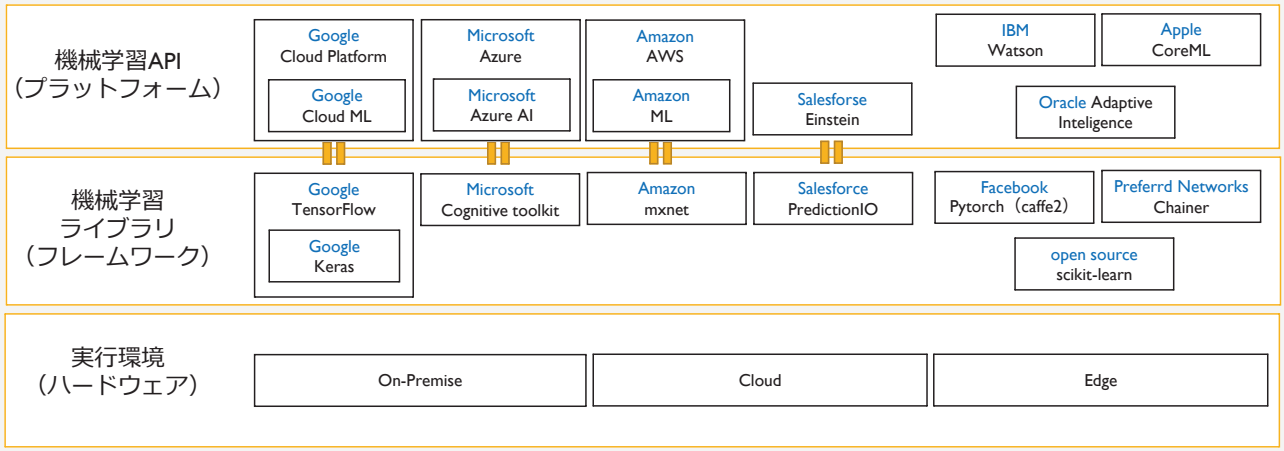
・演習：Google Colaboratory

機械学習を利用するには…

- 利用する方法は大きく4種類
  - AI PaaS/SaaS：ノンプログラミングで使えるクラウド上のAIサービス
  - 機械学習API：学習済みのモデルを外部プログラム（API）として使用
  - 機械学習ライブラリ：既存のライブラリを自身のプログラムで利用
  - スクラッチ開発：機械学習ライブラリを自作してイチから頑張る人



AI開発基盤 俯瞰図 2019



参考：気になるニュース＆ネット記事 - @IT
PyTorch vs. TensorFlow、ディープラーニングフレームワークはどっちを使うべきか問題
https://www.atmarkit.co.jp/ait/articles/1910/31/news028.html

2大機械学習プラットフォームの比較

Google Cloud Platform(GCP)

出典：「Cloud AI ビルディング ブロック | Cloud AI | Google Cloud」
https://cloud.google.com/products/ai/building-blocks/?hl=ja

Microsoft Azure AI

出典：「Azure クラウド サービス | Microsoft Azure」
https://azure.microsoft.com/ja-jp/services/#ai-machine-learning

今回使用する機械学習ライブラリ

scikit-learn（サイキットラーン）

- Pythonの機械学習ライブラリ
- オープンソース（BSD license）
- 多様なアルゴリズムを同じような記述で実装でき習得しやすい
- サンプルのデータセットも付属
- ただしGPUとディープラーニングに対応する予定はないらしい

TensorFlow（テンソルフロー）

- Google製の機械学習ライブラリ
- ディープラーニングをはじめとする最新の研究内容も取り入れ
- Kerasは、TensorFlowに対応した深層学習のラッパーライブラリ
- 2019年10月2日にKerasとの統合を強化した、TensorFlow2.0登場

Python言語を利用するには…

ローカル上で開発する（本格派）

- ケース①
  - Python（3.x系統）をインストール
  - 必要なライブラリをインストール
  - エディタでプログラムを記述する
  - pythonコマンドで翻訳・実行する
- ケース②
  - Anacondaをインストール。ライブラリも開発環境も一式込みで便利

クラウド上で開発する（お手軽）

- Google Colaboratory
  - インストール不要ですぐに使えるブラウザベースのPython開発環境
  - Googleが無償で提供。いくつかの制限はあるが機械・深層学習も可
  - Jupyterというオープンソースから派生（Anacondaにも付いている）
  - 本講座はColoboratoryで演習する

演習：Google Colaboratory

- Jupyter
  - 機械学習やデータ分析のコミュニティで広く使われている実行環境
  - セルと呼ばれる単位で実行が可能
  - 結果は即時、セル内に出力される
- インスタンス
  - Google上で動くLinuxの仮想マシン
  - Ubuntu系／マシン性能は随時進化
- Google Colaboratory
  - Jupyter+インスタンス ＝Colaboratory

Python（&Colaboratory）の基礎

2019/11/20 ver.
ここでは、データ分析を前提としたPythonの使い方を学びます。

すでに他の言語でプログラミング経験のある方には、わかりきっている内容もあると思いますが、Python独自の考え方やお作法もありますので、油断せずにトライしてください。

反面、プログラミング経験があまりない方にとっては、いきなりの演習で難しく感じるところもあると思いますが、まずは細かい文法などはあまり気にせずに「とりあえず、こんなことをしているんだな」といった概要を掴むように努力してみましょう。

開発環境の確認

今回の演習はすべて**Google Colaboratory**で行います。

Google Colaboratoryとは

Googleが無償で提供しているインストール不要ですぐに使えるブラウザベースのPython実行環境です。Jupyterというオープンソースから派生した実行環境がベースになっています。連続使用時間やファイルのアップロードに制限はありますが、機械学習や深層学習を行うことも可能です。

Pythonとは

Pythonとはプログラミング言語の１つで、以下のような特徴があります。

- 初心者向きで習得しやすい
- シンプルなコードで読みやすい
- 計算・統計処理で使用できるライブラリが豊富

人工知能やデータサイエンスに注目が集まる近年、開発・分析用のツールが豊富に用意されているPythonには急速に注目が集まっています。

Jupyterとは

機械学習やデータ分析のコミュニティで広く使われているレポート機能が付加されたプログラムの実行環境です。セルと呼ばれる単位でプログラムの実行が可能で、実行結果は即時、セルの下に出力されます。

ここで学習すること

- Colaboratory Tips

- Pythonの基本文法（変数・リスト）
- Numpyの基礎
- Pandasの基礎
- Pythonの基本文法（選択・繰り返し・関数）
- Matplotlibの基礎

Colaboratory Tips

ここでは、Google Colaboratoryを使う上で、知っていると便利な小技や裏技？をご紹介しながら、Jupyterベースの実行環境に慣れましょう。

マシンリソースを調べる

ColaboratoryはLinuxベースの仮想マシンで動作していますので、Linuxに詳しい方なら各種コマンドを使って様々なことを行うことができます。ここではGoogleが用意してくれた仮想マシンのリソースを調べてみましょう。

なお、Colaboratoryのセル内からLinuxのコマンドを実行する場合、行頭に「!」を付けてください。

```
# OS情報
! cat /etc/os-release
```

```
NAME="Ubuntu"
VERSION="18.04.3 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.3 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-
policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
```

注釈： セル内のコードを実行するには、いくつかの方法があります。

- セル左側の三角アイコンをクリックする
- Ctrl + Enterキーを押す（セルのフォーカスは移動しない）
- Shitf + Enterキーを押す（セルのフォーカスは下に移動する）
- 上部メニューの[ランタイム]から実行方法を選択する

```
# CPU
! cat /proc/cpuinfo
```

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 63
model name      : Intel(R) Xeon(R) CPU @ 2.30GHz
stepping        : 0
microcode       : 0x1
cpu MHz         : 2300.000
cache size      : 46080 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni
pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c
rdrand hypervisor lahf_lm abm invpcid_single ssbd ibrs ibpb stibp fsgsbase
tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt arat md_clear
arch_capabilities
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds
swapgs
bogomips        : 4600.00
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 63
model name      : Intel(R) Xeon(R) CPU @ 2.30GHz
stepping        : 0
microcode       : 0x1
cpu MHz         : 2300.000
cache size      : 46080 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 1
apicid          : 1
```

```
initial apicid  : 1
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni
pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c
rdrand hypervisor lahf_lm abm invpcid_single ssbd ibrs ibpb stibp fsgsbase
tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt arat md_clear
arch_capabilities
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds
swapgs
bogomips        : 4600.00
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

GPUの利用

```
# GPU
! nvidia-smi
```

```
NVIDIA-SMI has failed because it couldn't communicate with the NVIDIA driver. Make
sure that the latest NVIDIA driver is installed and running.
```

注釈: 初期状態では、GPUではなくCPUを利用しているので、上記のコードはエラーになります。

GPUを使用する場合は、Colaboratoryのメニューから[ランタイム]→[ランタイムのタイプを変更]で変更できます。[ハードウェアアクセラレータ]の項目を、NoneからGPUに変更して、もう一度、上記のセルを実行してみましょう。

時間制限（90分と12時間）

大変便利なColaboratory環境ですが、利用時間に90分と12時間という2種類の制限がある点は注意が必要です。

- 90分制限・・・インスタンス（仮想マシン）が落ちます。
  ⇒ つまり、変数に格納している情報や、importしたライブラリなどが全て忘れ去られます。
  （一旦ファイルとして保存したものは生き残ります。）

- 12時間制限・・・インスタンス（仮想マシン）が消えます。 ⇒ つまり、コードと実行結果の情報のみが残り、他は全て消えます。
  （一旦ファイルとして保存したものも全て消え、コード実行前の状態に戻ります。）

12時間制限に至るまでの残り時間をチェックするには、以下のコマンドを実行してください。

```
!cat /proc/uptime | awk '{print $1 /60 /60 "hours"}'
```

```
0.0686111hours
```

注釈:上記のコマンドでは、インスタンスが起動してからの経過時間が出力されます。この値が、12hoursになるまで、インスタンスが利用可能です。

セルの挿入と削除

Tipsの締めとして、セルを新たに挿入したり削除してみましょう。

セルの挿入は、画面上部の[＋コード] と [＋テキスト] で行えます。

- コード…Pythonを実行するためのセル（プログラム）
- テキスト…説明を記述するためのセル（Markdown記法）
    - 参考URL：Markdown記法 サンプル集 - Qiita

セルの削除は、セル右側のゴミ箱アイコンで行えます。

Pythonの基本文法（変数・リスト型）

ここではPythonの基本文法について演習を行います。と、いってもPythonの文法すべてを習得しようとすると膨大な時間がかかってしまいますので、ここでは今後の機械学習や深層学習の精度を左右するデータの前処理（データハンドリング）に必要となる要素に限定して演習をしてみましょう。

データ型を調べる（type）

Pythonに限らず、プログラミングのデータは データ型 によって分類されています。例えば、1 というデータは int型(整数型)、3.14は float型(浮動小数点数型) 、"abc" は str型(文字列型) というように分類されます。データ型はtype()の()の中にデータを入れることで、そのデータのデータ型を確認することができます。

```
# 整数の場合
type(1)
```

```
int
```

```
# 小数の場合
type(1.1)
```

```
float
```

```
# 文字の場合
type("abc")
```

```
str
```

変数の定義（ = ）

Pythonに限らずプログラミング言語全般で、データを保存しておくための 変数 というものが用意されています。変数は最初の文字がアルファベットか_であれば自由に名前をつけることができます。変数に値を代入するときには = の演算子を使います。

```
num = 3
```

```
moji = "abc"
```

type()の()の中に変数名を入れることで、その変数のデータ型を確認できます。

```
type(num)
```

```
int
```

```
type(moji)
```

```
str
```

print()の()の中に変数名をいれると、変数に代入されたデータを表示することができます。

```

```

```
print(num)
print(moji)
```

```
3
abc
```

printと同じような機能に、displayがあります。printはpython標準の命令（関数とも呼びます）ですが、displayはJupyterの機能になります。出力の違いを確認しつつ、今後は好みで使い分けてください。

TIPS: displayは後述するDataFrameの形を崩さずに出力できるため重宝します。

```
display(num)
display(moji)
```

```
3
```

```
'abc'
```

注釈: Jupyterでは、わざわざprintを使用しなくても、セルの最終行に変数名を記述すると、変数の内容を表示することができます。ただし、セルの最終行に書いたもののみ有効ですので注意してください。

```
num
```

```
3
```

```
moji
```

```
'abc'
```

```
# この場合、最終行のみ有効で、1行目は無効となる
num
moji
```

```
'abc'
```

```
# 2つ以上の変数の内容を表示したい場合は、printかdisplayを使う
display( num )
display( moji )
```

```
3
```

```
'abc'
```

リスト型（list）

先ほどは変数にデータを保存しましたが、一つの変数に複数のデータを入れたい場合があります。そのときに使うのが リスト型（list)になります。

```
# リスト型の定義の仕方
lst = [10, 20, 30, 40, 50]
```

リストの中の各データを要素と呼びます。最初の要素には0、次の要素には1というように各要素にはインデックス（添字）と呼ばれる番号が自動的に振られています（インデックスの最初が0番から始まることに注意してください）そのインデックスを指定することで、リストの要素を取り出すことができます。

```
# リストから3番目の要素を取り出す(3番目の要素の添字は2になる)
lst[2]
```

```
30
```

listの後半の要素を取得したい場合などは、-(マイナス) を使ってインデックスを指定することで、最後から○番目の値を取得することができます。

```
lst[-1]
```

```
50
```

また、スライシング（slicing）と呼ばれる方法で、複数のデータを取り出してくることができます。

```
# 1番目から3番目までのデータを取り出す
lst[1: 4]
```

```
[20, 30, 40]
```

注釈： スライシングで範囲指定した際に、左端の要素は含みますが、右端の要素は含まれないことに注意してください。

```
# 最初から4番目の要素まで取り出す
lst[: 4]
```

```
[10, 20, 30, 40]
```

```
# 3番目の要素から最後の要素まで取り出す
lst[2: ]
```

```
[30, 40, 50]
```

Numpyの基礎

PythonにはNumpyという便利なライブラリがあります。 Numpyを利用すると、Python標準のリスト型に比べて、多次元配列のデータを効率よく扱うことができます。また、Numpyは標準偏差や分散といった統計量を出力してくれる関数が用意されており、科学技術計算の基盤となっています。

Numpyを利用するには、まず次のようにインポートを行います。

```
# Numpyライブラリを、npという別名で使用できるように宣言する
import numpy as np
```

numpy.ndarray

Numpyには、ndarrayと呼ばれるデータ型が用意されています。 ndarrayは一見すると、Python標準のリスト型（list）と似ていますが、ベクトルや行列のように、内部の要素を一括して計算できる便利な機能があります。

```
# arrayを定義する
ary = np.array([1, 2, 3])
display( type(ary) )
display( ary )
```

```
numpy.ndarray



array([1, 2, 3])
```

```
# aryのすべての要素に5をかける
ary * 5
```

```
array([ 5, 10, 15])
```

ちなみに同じことをPython標準のリスト型でやろうとすると、全く結果が異なることを確認してみましょう。

```
# リスト型の定義の仕方
lst = [1, 2, 3 ]
lst * 5
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

なお、前述の ary * 5 では、ary内部の値を更新していませんので、もう一度ary の中身を表示させると、[5,10,15]ではなく、[1,2,3]のままなのが確認できます。

```
ary
```

```
array([1, 2, 3])
```

ary内部の値を更新するには、以下のように = を使って、新たな値を代入してあげる必要があります。この「値を更新したつもりで、実はしていなかった！」というミスは、プログラミングに慣れるまで（むしろ慣れていても）結構やりがちな失敗なので気をつけましょう。

```
ary = ary * 5
ary
```

```
array([ 5, 10, 15])
```

numpyでの基礎統計量

通常、標準偏差などを計算したい場合は自分で実装する必要がありますが、Numpyは基本的な統計量を計算する関数を用意しています。

```
# range関数で、0～9までの連続した整数値を生成する
ary = np.array(range(10))
ary
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
# aryの平均値
ary.mean()
```

```
4.5
```

```
# aryの分散
ary.var()
```

```
8.25
```

```
# aryの標準偏差
ary.std()
```

```
2.8722813232690143
```

```
# aryの最小値
ary.min()
```

```
0
```

```
# aryの最大値
ary.max()
```

```
9
```

Pandasの基礎

Pandasは、Pythonでのデータ分析ライブラリとして最も活用されているライブラリのひとつです。PandasはNumpyを基盤に、シリーズ（Series）とデータフレーム（DataFrame）というデータ型を提供します。

Pandasを利用するには、まず次のようにインポートを行います。

```
# Pandasライブラリを、pdという別名で使用できるように宣言する
import pandas as pd
```

pandas.Series

ここまで、Python標準のリスト型（list）と、Numpyライブラリのndarrayを扱いましたが、Pandasライブラリにも同様の機能を有するシリーズ（Series）というデータ型があります。

```
srs = pd.Series([10, 20, 30, 40, 50])
srs
```

```
0    10
1    20
2    30
3    40
```

```
4    50
dtype: int64
```

共通点

```
# list/ndarrayとの共通点① 添字を指定して任意の要素を取り出せる
srs.iloc[3]
```

```
40
```

```
# list/ndarrayとの共通点② スライシングが使える
srs.iloc[ 1 : 3 ]
```

```
1    20
2    30
dtype: int64
```

相違点

```
# list/arrayと相違点① indexを指定できる
srs = pd.Series([10, 20, 30, 40, 50], index=["one", "two", "three", "four",
"five"])
srs
```

```
one      10
two      20
three    30
four     40
five     50
dtype: int64
```

```
# list/ndarrayと相違点② np.arrayよりも便利な機能（メソッド）が充実している ※詳細は後ほど
# 要約統計量を出力する
srs.describe()
```

```
count     5.000000
mean     30.000000
std      15.811388
min      10.000000
25%      20.000000
50%      30.000000
75%      40.000000
max      50.000000
dtype: float64
```

注釈: NumpyとPandasの使い分けについて

NumpyはPythonで弱点とされてきた計算速度の遅さを克服しています。PandasはNumpyを利用して使いやすくしたライブラリです。 したがって、一般的にはPandasの方がNumpyを呼び出すオーバーヘッドが発生する分、直接、Numpyを利用したほうがパフォーマンスは上がりやすい傾向にあります。

数値演算の処理速度を最適化したい場合には、Pandasだけでは難しいかもしれません。ただし、Pnadas（特にこれからお話するDataFrame）は速度の面に若干目をつぶっても、余りある利点（使いやすさと機能の豊富さ）があります。

ですので、まずはPandasでトライしてみて、どうしても処理速度に問題があるようでしたら、NumpyやSQLを利用するという切り替えもありかと思います。

pandas.DataFrame

Pandasライブラリで最も使われるデータ型にデータフレーム（DataFrame）があります。Seriesは1次元のデータしか扱えませんが、DataFrameは行列のような多次元のデータも扱えます。

####DataFrameの生成
まずはサンプルデータを含んだ、DataFrameを生成してみます。

```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	0	1	2
0	1	2	3

	0	1	2
1	4	5	6
2	7	8	9

indexとcolumnの指定

DataFrameはインデックス（index）やカラム（column）を指定することにより、Excelのシートのようにデータを扱うことができます。

```
df = pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]], index=["a", "b", "c"], columns=["A",
"B", "C"])
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	A	B	C
a	1	2	3
b	4	5	6
c	7	8	9

indexとcolumn名の変更

pandas.DataFrame.renameを使うことで、行や列の名前を変更することが出来ます。

```
df.rename(columns={"A":"a"},index={"a":"A"})
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	a	B	C
A	1	2	3
b	4	5	6
c	7	8	9

注釈: DataFrameを扱うときに多いミスが「更新したつもり、だけど更新されてなかった」系の失敗です。実は上記のrenameは、その場限りのもので、もう一度、dfの中身を表示すると、更新されておらず困惑します（慣れている人でも、かなりありがちなミスです）

```
# 変更が反映されておらず、元に戻っている！
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	A	B	C
a	1	2	3
b	4	5	6
c	7	8	9

それを解決するには、主に２つの方法があります。

```
# inplaceオプションををtrueにする
df.rename(columns={"A": "a"},index={"a": "A"}, inplace=True)
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	a	B	C
A	1	2	3
b	4	5	6
c	7	8	9

```
# イコール( = )で新しい内容を同じdfに代入する
df  = df.rename(columns={"C": "c"},index={"c": "C"} )
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	a	B	c
A	1	2	3
b	4	5	6
C	7	8	9

データの消去

pandas.Series.dropは、列と行の削除を行うことが出来ます。デフォルトでは、行の削除(axis=0)を指定しています。

```
df.drop('C')
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	a	B	c
A	1	2	3
b	4	5	6
C	7	8	9

注釈:おっと、C行が削除されていませんね…　そうです、先ほどご紹介した「かなりありがちなミス」です。inplace=Trueオプションを付けて試してみましょう。

```
# inplaceオプションで、削除を反映させる
df.drop('C' , inplace=True)
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	a	B	c
A	1	2	3
b	4	5	6

```
# axis=1を指定することで、列を削除することが出来る。
df.drop('c', axis=1, inplace=True)
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	a	B

	a	B
A	1	2
b	4	5

データの取得

pd.DataFrame.locとpd.DataFrame.ilocを使うことで、DataFrameの指定した行や指定した列を取得することができます。

```
# データフレームを再度作成する
df = pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]], index=["a", "b", "c"], columns=["A",
"B", "C"])
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	A	B	C
a	1	2	3
b	4	5	6
c	7	8	9

```
# b行のデータを取得
df.loc["b"]
```

```
A    4
B    5
C    6
Name: b, dtype: int64
```

```
# C列のデータを取得
df.loc[ : , "C"]
```

```
a    3
b    6
c    9
Name: C, dtype: int64
```

```
# 0,1行目のデータを取得
df.iloc[ [0, 1] ]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	A	B	C
a	1	2	3
b	4	5	6

```
# 1列目のデータを取得
df.iloc[ : , [1]]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	B
a	2
b	5
c	8

データの表示

Dataframeにはheadとtailというメソッドが用意されています。headは先頭から指定した数の行データを表示します。tailは、末尾から指定した数の行データを表示します。

```
# head
df.head(2)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	A	B	C
a	1	2	3
b	4	5	6

```
# tail
df.tail(2)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	A	B	C
b	4	5	6
c	7	8	9

注釈:大量のデータを扱う場合、「まずは数行だけ…」を分析対象にしたい場合など、head(tail)は重宝しますので、覚えておきましょう。

条件に応じた行の抽出

Dataframeにはqueryというメソッドが用意されています。文字列で条件をしてすることで、条件を満たした行だけを取り出すことができます。SQLに詳しい方は、SELECT～WHERE句をイメージすると分かりやすいかもしれません。

```
# データフレームの中身を確認
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	A	B	C
a	1	2	3
b	4	5	6
c	7	8	9

```
# B列が5の行を抽出
df.query('B == 5')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	A	B	C
b	4	5	6

```
# B列が5以上の行を抽出
df.query('B >= 5')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	A	B	C
b	4	5	6
c	7	8	9

```
# B列が5以上で、かつC列が7以上の行を抽出
df.query('B >= 5 & C >=7')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	A	B	C
c	7	8	9

```
# A列が1、またはC列が9の行を抽出
df.query('A == 1 | C ==9')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	A	B	C
a	1	2	3
c	7	8	9

Pythonの基本文法（選択・繰り返し・関数）

今回の演習ではあまり使うことはありませんが、少し複雑な前処理や処理手順を組もうとした場合、選択（if）や、繰り返し（for）といった構文を使用します。

現段階で理解する必要はありませんが、他のプログラム言語を習得している方向けに、Pythonでのifやforの記述法をご紹介します。

選択（if文）

if 文の使い方は以下の通りです。例えばJava言語では{ }で囲んでいた部分が、Pythonでは字下げによって行われていることに注意してください。

if 条件:
　　実行したい文
　　※ifの中の行は、必ずインテント（字下げ）が必要です

インテントは半角スペース4個が推奨されていますが、任意の個数やTabでも正しく動きます（可読性を考慮すると、せめて社内やプロジェクト単位では統一すべきだと思いますが…）

```
lang = "python"
if lang == "python":
    # インテントされた部分がifの中身として認識される
    print("I love Python!")
    print("I love Programming!")
```

```
I love Python!
I love Programming!
```

```
lang = "java"
if lang == "python":
    # インテントされた部分がifの中身として認識される
    print("I love Python!")
    print("I love Programming!")
```

注釈:上記は何も表示されません。これは、laung が "ruby"になったため、条件を満たしていないためです。

```

```

```
lang = "java"
if lang == "python":
    # インテントされた部分がifの中身として認識される
    print("I love Python!")
print("I love Programming!")
```

```
I love Programming!
```

注釈:上記の例では、インテントしていない"I love Programming!"は、if文の動作とは全く関係なく実行されることになります。

ところで、lang が "python" 以外のときにもなにか実行したい場合があるとしましょう。そんなときに使うのが　else文 です。else 文は、if文の条件を満たさなかった場合に実行したい文を書くことができます。
else文の使い方は、

if 条件:
　　実行したい文

else:
　　実行したい文

```
lang = "ruby"
if lang == "python":
    print("I love python!")
else:
    print("I love other language!")
```

```
I love other language!
```

繰り返し(for文)

同じ処理を繰り返したい場合は、for文を使用します。他の言語に比べると、繰り返し条件の記述に独特の部分があるので、以下を実行例を参考にしてください。

for 変数 in リスト型:
　　繰り返したいしたい文
　　※ifと同様に、for中の行は、必ずインテント（字下げ）が必要です

```
# xを0から4まで増分させながら繰り返す
for x in [0, 1, 2, 3, 4]:
    print(x)
```

```
0
1
2
3
4
```

上記のような繰り返し条件の指定では、例えば0から100までの値を実行したいような場合に、リスト型で100まで定義するのは現実的ではありません。そのような際に使うのがrangeです。rangeの使い方は以下の通りです。

```
# xを0から4まで増分させながら繰り返す
for x in range(5):
    print(x)
```

```
0
1
2
3
4
```

```
# xを2から4まで増分させながら繰り返す
for x in range(2, 5):
    print(x)
```

```
2
3
4
```

```
# xを0から9まで2ずつ増分させながら繰り返す
for x in range(0, 9, 2):
    print(x)
```

```
0
2
4
6
8
```

参考：for文とDataFrame

著者の経験ですが、for文で、DataFrameから1行ずつ行を取り出して、その1行に対して何か処理を行いたいときが結構あります。この段階で覚える必要はありませんが、何かの際にここに書いてあったことを思い出してください。ただし、何万件もの行をfor文で取り出そうとすると、滅茶苦茶な処理時間がかかりますので、まずはqueryメソッドなどで取り出す件数を絞ってから、for文に与えるようにしましょう。

```
# データフレームを作成する
# サンプルデータ：気温(℃)とアイスクリームの売上(個)の関係
df = pd.DataFrame([ [10,20],[15,30],[15,40],[25,70],[30,100] ], columns=["気温",
"売上"])
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	気温	売上
0	10	20
1	15	30
2	15	40
3	25	70
4	30	100

```
# データフレームから一行ずつ取り出す
for index,row in df.iterrows():
    kion   = row['気温']
    uriage = row['売上']
    print( kion )
    print( uriage )
```

```
10
20
15
30
```

```
15
40
25
70
30
100
```

```
# おまけ：上記のprintをformatメソッドで見やすくする
for index,row in df.iterrows():
    kion   = row['気温']
    uriage = row['売上']
    print( '気温が{}℃の日は、アイスクリームの売上は{}個でした'.format( kion , uriage
) )
```

```
気温が10℃の日は、アイスクリームの売上は20個でした
気温が15℃の日は、アイスクリームの売上は30個でした
気温が15℃の日は、アイスクリームの売上は40個でした
気温が25℃の日は、アイスクリームの売上は70個でした
気温が30℃の日は、アイスクリームの売上は100個でした
```

関数の定義（def）

Pythonに限らず、多くのプログラミング言語に関数（サブルーチン、メソッドとも）というものがあります。関数は数学で扱うように「何かの値を入れると、何らかの値を出力する」という役目がありますが、プログラミングの場合には、それと同時に「いくつかのまとまった処理をひとまとめにする」という役割もあります。Pythonでの関数の定義の方法は以下の通りです。

def 関数名(仮引数):
    実行したい文1
    実行したい文2
    実行したい文3
    return 出力したい値

```
# x * y を 計算する関数を定義する（関数名:calc）
def calc( x , y ):
    result = x * y
    return result
```

注釈:上記のセルでは、関数を定義しただけですので、実行しても何も出力されません。関数を動作させるには、この関数に対して、xとyの値を渡して実行する必要があります。なお、厳密にはこのとき関数を定義する際に()内に設定する変数を仮引数、 関数を実行する際に入れる実際の値を引数といって区別します（が、そこまでこだわらなくても今は大丈夫です）

```
# 関数calcを実行する（引数は4と5）
calc(4,5)
```

```
20
```

関数からの出力（これを戻り値と呼びます。関数の最後でreturn文により指定された値です）は、変数に格納することもできます。

```
# 戻り値を変数（kakezan1と2）に代入する
kakezan1 = calc(5,5)
kakezan2 = calc(10,10)

print( kakezan1 + kakezan2 )
```

```
125
```

Matplotlibの基礎

データを分析する際に、値を見るだけでは全体像を捉えにくいので、まずはグラフで描画して俯瞰することが多いです。ここでは、Pythonでよく利用されるグラフ描画のライブラリであるMatplotlibの簡単な使い方をご紹介します。

```
# Matplotlibライブラリを、pltという別名で使用できるように宣言する
import matplotlib.pyplot as plt
```

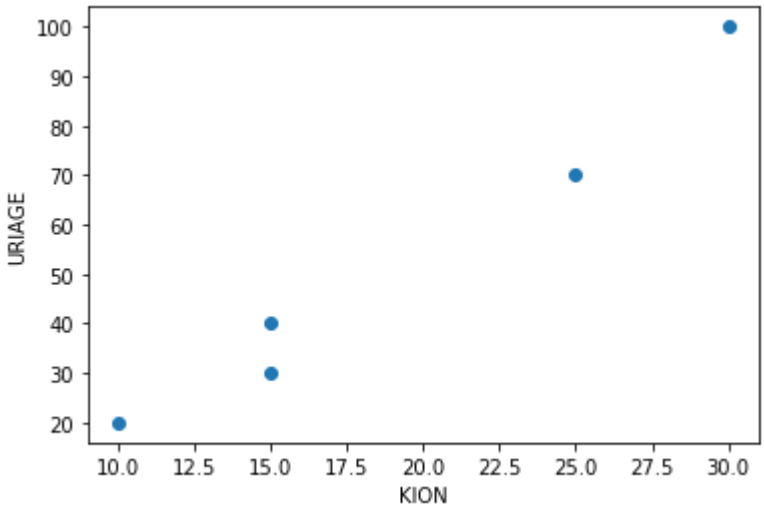
```
# データフレームを作成する
# サンプルデータ：気温(℃)とアイスクリームの売上(個)の関係
df = pd.DataFrame([ [10,20],[15,30],[15,40],[25,70],[30,100] ], columns=["気温",
"売上"])
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

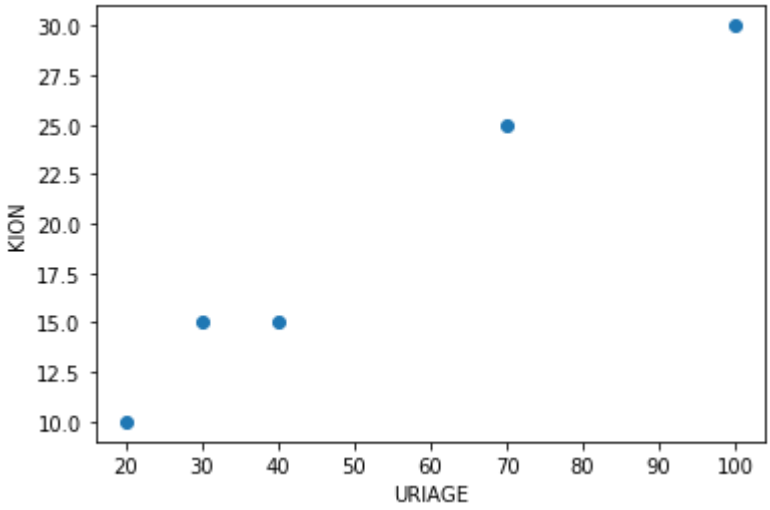
	気温	売上
0	10	20
1	15	30
2	15	40
3	25	70
4	30	100

散布図

```
# 散布図1
plt.figure()
plt.scatter(x=df['気温'], y=df['売上'])
plt.xlabel('KION')
plt.ylabel('URIAGE')
plt.show()
```



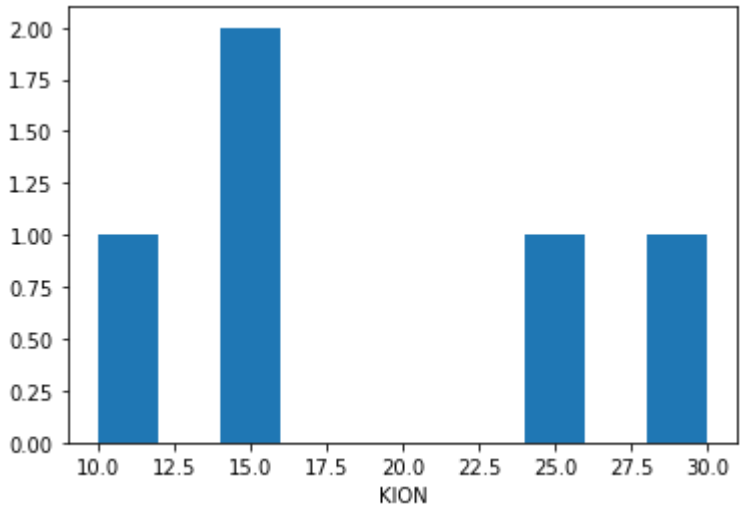

```
# 散布図2
plt.figure()
plt.scatter(x=df['売上'], y=df['気温'])
plt.xlabel('URIAGE')
plt.ylabel('KION')
plt.show()
```



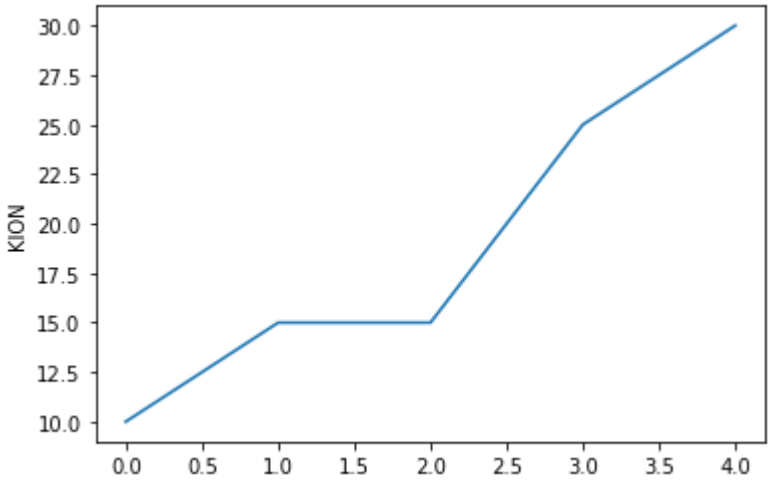
様々なグラフ

Matplotlibは、散布図の他にも、ヒストグラム、折れ線、箱ひげ図、ヒートマップなど様々な様式のグラフに対応しています。また、オプションも数多く用意されていますが、ここではいくつかグラフの種類を変えて試してみましょう（あくまでグラフ描画の練習ですので、グラフとしてあまり意味をなしていないものもありますが、ご了承ください）

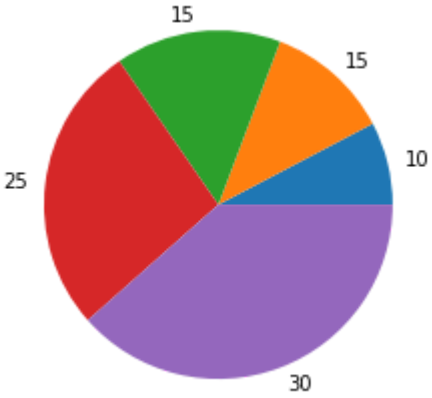
```
# ヒストグラム
plt.figure()
plt.hist(df['気温'])
plt.xlabel('KION')
plt.show()
```




```
# 折れ線
plt.figure()
plt.plot(df['気温'])
plt.ylabel('KION')
plt.show()
```




```
# 円グラフ
plt.figure()
plt.pie(df['売上'],labels=df['気温'])
plt.show()
```



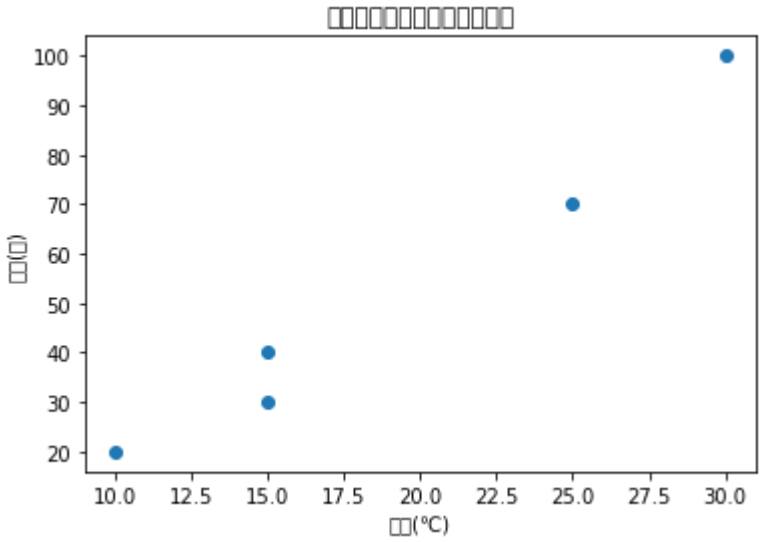
参考：文字化け□□対策

Colabratory環境でMatplotlibを使用すると、ラベルの部分の全角文字が文字化け□□になってしまいます。そのため、基本的には講義では半角文字でラベル付けをしていきますが、「どうしても日本語がいい！」という人のために、Mini Tipsとして、Matplotlibの日本語対応の方法についてご紹介します。

```
# 文字化けしてしまう例
plt.figure()
plt.scatter(x=df['気温'], y=df['売上'])
plt.title('気温とアイスクリームの関係')
plt.xlabel('気温(℃)')
plt.ylabel('売上(個)')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 27671 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 28201 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 12392 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 12450 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 12452 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 12473 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 12463 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 12522 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 12540 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 12512 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 12398 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 38306 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 20418 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 27671 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 28201 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 22770 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 19978 missing from current font.
```

```
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:211:
RuntimeWarning: Glyph 20491 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 22770 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 19978 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 20491 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 12392 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 12450 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 12452 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 12473 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 12463 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 12522 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 12540 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 12512 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 12398 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 38306 missing from current font.
  font.set_text(s, 0, flags=flags)
/usr/local/lib/python3.6/dist-packages/matplotlib/backends/backend_agg.py:180:
RuntimeWarning: Glyph 20418 missing from current font.
  font.set_text(s, 0, flags=flags)
```



注釈:たくさんのRuntimeWaarningが出力されて、実行結果は□□だらけになることが確認できます。

これを回避するには、いくつかの段階を踏む必要があります（少し手間がかかります）

まず、日本語用のフォントを apt-getコマンドでダウンロードして、不要なキャッシュを削除します。

```
# 日本語フォントをダウンロードする。
! apt-get -y install fonts-ipafont-gothic

# キャッシュを削除する。
# 削除すべきキャッシュのファイル名は、Matplotlibがバージョンアップすると変わるため、
# 下記のrmでうまく行かない場合、! ls -ll /root/.cache/matplotlib/ でファイル名を確認
# 旧ファイル名: ! rm /root/.cache/matplotlib/fontList.json
# 旧ファイル名: ! rm /root/.cache/matplotlib/fontlist-v300.json

! rm /root/.cache/matplotlib/fontlist-v310.json    # 2019/10/31段階でのファイル名
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
fonts-ipafont-gothic is already the newest version (00303-18ubuntu1).
The following package was automatically installed and is no longer required:
  libnvidia-common-430
Use 'apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 7 not upgraded.
```

注釈:ダウンロードのバーが何回か出力された後に、処理が停止したら、次のステップへ進みます（20～30秒程度）

次は日本語フォントをColaboratoryに反映させるため、以下の手順でランタイムを再起動します。

- Colaboraryの上部メニュー → [ランタイム] → [ランタイムを再起動]

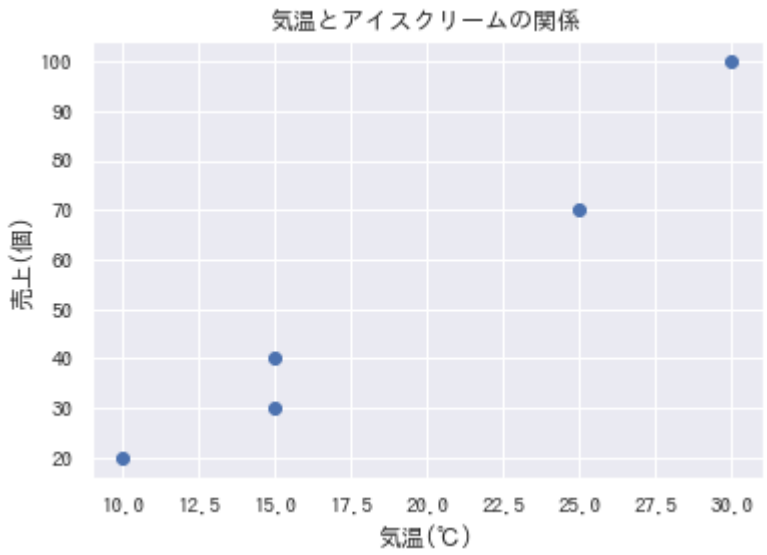
画面の右上が一瞬「再起動中」→「初期化中」となり、その後に「RAM」「ディスク」のバーが表示されたら、再起動は完了です（5秒程度）

続いて、以降のセルを実行してグラフを描画します。なお、ランタイムを再起動しましたので、もう一度、各ライブラリのimportと、DataFrameの生成を行う必要があります。

```
# ライブラリをインポートする
import pandas as pd
import matplotlib.pyplot as plt

# データフレームを作成する
# サンプルデータ：気温(℃)とアイスクリームの売上(個)の関係
df = pd.DataFrame([ [10,20],[15,30],[15,40],[25,70],[30,100] ], columns=["気温",
"売上"])
```

```
# 日本語フォントの設定
import seaborn as sns
sns.set(font='IPAGothic')
```

```
plt.figure()
plt.scatter(x=df['気温'], y=df['売上'])
plt.title('気温とアイスクリームの関係')
plt.xlabel('気温(℃)')
plt.ylabel('売上(個)')
plt.show()
```



注釈:お疲れ様でした。以上、少し手間はかかりますが、見やすくなりました。日本語化したい場合は、この一連の流れをプログラム冒頭のセルで「おまじない」的に実行してしまうのも手かもしれません。

まとめ

ここでは、以下のことを学習しました。

- Colaboratory Tips
- Pythonの基本文法（変数・リスト）
- Numpyの基礎
- Pandasの基礎
- Pythonの基本文法（選択・繰り返し・関数）
- Matplotlibの基礎

AI2-2

AIの実装（回帰／分類）

－ 講義内容 －

- 講師 自己紹介
- 回帰と分類の違い
- 演習：ディープラーニングによる回帰
- 演習：ディープラーニングによる分類

この住宅はいくらで売れる？（回帰）

有名なボストンの住宅価格データセットを使って、住宅がいくらで売れるか予測してみましょう。
このデータセットは、1978年にきれいな空気と住宅の価格に関する論文で使用されたものです。
米国ボストン市郊外における地域別の住宅価格が、町の犯罪率や住宅の部屋の数など様々な情報とセットになっています。
このデータセットを使って、機械学習の流れを体験しましょう。

ここで学習すること

- 教師ありの機械学習の流れ
- データの前処理
- ニューラルネットワークモデルの作り方、使い方
- 回帰モデルの評価方法
- 作成したモデルの保存方法、保存したモデルの使い方

回帰問題とは

データをもとに、目的となる数値を予測する問題を「回帰問題」といいます。
例）
・フリマアプリの過去の出品データをもとに、落札される金額を予測する。
・過去の気温や天候データをもとに、気温を予測する。

今回は、環境など複数のデータをもとに住宅の価格を予測する「重回帰」を扱います。

教師ありの機械学習の流れ

教師ありの機械学習のプロジェクトは、以下のように進んでいきます。

1. プロジェクトの目標を決める
   ビジネスで解決したい課題と、精度の目標を決めます。
2. データを収集する
   必要なデータを収集し、ラベルやアノテーションを付加します。
3. データを分析する
   収集したデータを分析し、相関を見出したり、加工が必要かどうか検討します。
4. データを前処理する
   機械学習に適した形にデータを加工します。
   欠損値を代替値に置換したり、ノイズになるデータを除外します。
   データによっては、特徴を強く出すための加工を行います。
   また、機械学習を進めやすいよう正規化を行います。
   データを訓練データ、テスト（評価）データに分割します。
5. モデルを定義する
   モデルの構造を定義します。
6. 学習する
   訓練データを使って学習します。
7. 評価する
   テストデータを使って、モデルの精度を求めます。
8. チューニング
   目標の精度に達するように、データやモデルの構造、ハイパーパラメータを調整します。
   チューニング→学習→評価を繰り返します。
9. モデルを保存する 作成したモデルをファイルに保存します。
   保存したモデルはいつでも使うことができます。
10. 推論
    モデルを使って、未知のデータで推論を行います。
11. 再学習
    運用で新たに追加されていくデータを使って、モデルの再学習を行います。

目標の設定

目的に合わせて、モデルの精度の目標を設定します。
例えば、このモデルの目的は人間が住宅の売値を決める時の参考値を得ることで、実用可能な誤差（MAE）は2,000ドル以下、という風に設定します。

回帰モデルの性能を表す指標として、予測値が正解の値から平均でどれだけ離れているかを表す「平均絶対誤差（MAE）」がよく用いられます。

もし、売値をシステムで自動決定することを目的とするなら、目標の誤差はもっと小さい値を設定しないといけないでしょう。

データの収集

本来はここでデータを収集してラベル付けを行いますが、今回は既存のデータセットを使います。

ライブラリのインポート

プログラムで使用するライブラリをインポートします。
今回は以下のライブラリを使用します。

- numpy … NumPy(https://numpy.org/) 数値計算に使用します。
- pandas … pandas(https://pandas.pydata.org/) データの操作・統計に使用します。
- matplotlib … matplotlib(https://matplotlib.org/) グラフの描画に使用します。
- seaborn … seaborn(https://seaborn.pydata.org/) matplotlibのラッパーで、便利なグラフ描画機能が揃っています。
- tensorflow … TensorFlow(https://www.tensorflow.org/?hl=ja) ニューラルネットワークを使用するためのフレームワークです。
- tensorflow.keras … Keras(https://keras.io/ja/) TensorFlowのラッパーです。
- sklearn … scikit-learn(https://scikit-learn.org/stable/) 機械学習のライブラリです。

```
# matplotlibの最新バージョンを使用する
!pip install matplotlib --upgrade
```

```
Requirement already up-to-date: matplotlib in /usr/local/lib/python3.6/dist-packages (3.1.2)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from
matplotlib) (2.6.1)
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from
matplotlib) (1.1.0)
Requirement already satisfied, skipping upgrade: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from
matplotlib) (0.10.0)
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.6/dist-packages (from matplotlib) (2.4.5)
Requirement already satisfied, skipping upgrade: numpy>=1.11 in /usr/local/lib/python3.6/dist-packages (from matplotlib)
(1.17.4)
Requirement already satisfied, skipping upgrade: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-
dateutil>=2.1->matplotlib) (1.12.0)
Requirement already satisfied, skipping upgrade: setuptools in /usr/local/lib/python3.6/dist-packages (from
kiwisolver>=1.0.1->matplotlib) (41.6.0)
```

```
# ライブラリのインポート
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# TensorFlow 2.x系を使う
%tensorflow_version 2.x
import tensorflow as tf
from tensorflow.keras import models, layers, regularizers
from tensorflow.keras.datasets import boston_housing

# TensorFlowのバージョン
print(tf.__version__)

from sklearn.model_selection import train_test_split

# matplotlib の出力をノートブック上に描画するための設定
%matplotlib inline
```

```
TensorFlow 2.x selected.
2.0.0
```

データの読み込み

使用するデータセットを読み込みましょう。
tf.kerasの機能を使ってボストンの住宅価格データセットを読み込みます。
ここで訓練データとテストデータの割合を決めることができます。
2割をテストデータとして使うことにしましょう。

```
# データを読み込み
(x_train, y_train), (x_test, y_test) = boston_housing.load_data(test_split=0.2, seed=777)
```

`load_data()` の戻り地は以下のとおりです。

- x_train … 訓練データ
- y_train … 訓練データのラベル
- x_test … テストデータ
- y_test … テストデータのラベル

データの分析

まずはデータの形式を確認しましょう。

データの構成

データ x には以下の値が格納されています。

カラム名	内容
CRIM	一人当たりの犯罪率
ZN	25,000平方フィートを超える敷地に区画された宅地の割合
INDUS	非小売業の土地面積の割合
CHAS	チャールズ川ダミー変数。川に面している場合は1、それ以外の場合は0
NOX	一酸化窒素濃度（1000万分の1）
RM	1住戸あたりの平均部屋数
AGE	1940年以前に建設された住宅の割合
DIS	5つのボストンの雇用センターまでの加重距離
RAD	ラジアルハイウェイへのアクセスのしやすさの指標
TAX	10,000ドルあたりの固定資産税率
PTRATIO	生徒教師比率
B	黒人の割合（1000（Bk - 0.63）^ 2）
LSTAT	低所得者の割合

ラベル y には、住宅の価格が格納されています。

データの型、行数、列数

```
print('データ型')
print('x type:', type(x_train), x_train.dtype)
print('y type:', type(y_train), y_train.dtype)
print('x_test type:', type(x_test), x_test.dtype)
print('y_test type:', type(y_test), y_test.dtype)

print('\n行数, 列数')
print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)
print('x_test shape:', x_test.shape)
print('y_test shape:', y_test.shape)
```

```
データ型
x type: <class 'numpy.ndarray'> float64
y type: <class 'numpy.ndarray'> float64
x_test type: <class 'numpy.ndarray'> float64
y_test type: <class 'numpy.ndarray'> float64

行数, 列数
x_train shape: (404, 13)
y_train shape: (404,)
```

```
x_test shape: (102, 13)
y_test shape: (102,)
```

pandas.DataFrame に変換する

データを扱いやすいよう、2次元の表形式のデータを扱う `pandas.DataFrame`に変換しましょう。

```
# 列名を定義
columns_name = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
target_column_name = ['target']

# DataFrameに変換
x_train_df = pd.DataFrame(x_train, columns=columns_name)
y_train_df = pd.DataFrame(y_train, columns=target_column_name)
x_test_df = pd.DataFrame(x_test, columns=columns_name)
y_test_df = pd.DataFrame(y_test, columns=target_column_name)
```

データの中身を見てみましょう。

```
# pandasの表示の形式を指定
pd.options.display.float_format = '{:.2f}'.format

# 訓練データの先頭
x_train_df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.25	0.00	10.59	0.00	0.49	5.78	72.70	4.35	4.00	277.00	18.60	389.43	18.06
1	15.86	0.00	18.10	0.00	0.68	5.90	95.40	1.91	24.00	666.00	20.20	7.68	24.39
2	7.40	0.00	18.10	0.00	0.60	5.62	97.90	1.45	24.00	666.00	20.20	314.64	26.40
3	0.13	25.00	5.13	0.00	0.45	6.76	43.40	7.98	8.00	284.00	19.70	395.58	9.50
4	0.04	80.00	4.95	0.00	0.41	6.86	27.90	5.12	4.00	245.00	19.20	396.90	3.33

```
# ラベルの先頭
y_train_df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	target
0	22.50
1	8.30
2	17.20

	target
3	25.00
4	28.50

欠損値対応

訓練データに値の欠損があるか調べましょう。

```
# 値の欠損の数
x_train_df.isna().sum()
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
dtype: int64
```

値の欠損はありませんでした。
もし欠損がある場合は、0などの代替値に置換しておきます。

基本統計

訓練データの基本統計を出力してみましょう。
基本統計は、DataFrameの `describe()` で出力できます。
出力される項目の意味は以下のとおりです。

項目	意味
count	データの個数
mean	平均値
std	標準偏差
min	最小値
25%	1/4分位数
50%	中央値
75%	3/4分位数
max	最大値

```
# 訓練データの基本統計
x_train_df.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
count	404.00	404.00	404.00	404.00	404.00	404.00	404.00	404.00	404.00	404.00	404.00	404.00	404.00
mean	3.99	11.35	11.43	0.08	0.56	6.26	69.59	3.79	9.78	414.15	18.44	353.87	13.02
std	9.33	23.66	6.94	0.27	0.12	0.71	28.04	2.15	8.87	171.11	2.18	95.29	7.29
min	0.01	0.00	0.74	0.00	0.39	3.86	2.90	1.13	1.00	187.00	12.60	0.32	1.73
25%	0.08	0.00	5.19	0.00	0.45	5.87	45.55	2.06	4.00	279.00	17.00	374.71	7.38
50%	0.27	0.00	9.90	0.00	0.54	6.18	79.20	3.10	5.00	335.00	19.05	391.18	11.71
75%	4.13	12.50	18.10	0.00	0.63	6.62	94.53	5.23	24.00	666.00	20.20	396.21	17.17
max	88.98	100.00	27.74	1.00	0.87	8.78	100.00	12.13	24.00	711.00	22.00	396.90	37.97

CHAS には 0 と 1 が含まれますが、ほとんどが 0 であることなどが読み取れます。

ラベルの値を見てみましょう。
y_train には住宅価格（単位はKドル。ちなみに1978年は1ドル152円でした）が格納されています。

```
# 訓練データのラベルの基本統計
y_train_df.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	target
count	404.00
mean	22.21
std	9.22
min	5.00
25%	16.58
50%	20.95
75%	24.80
max	50.00

住宅価格は5,600ドル～50,000ドル（約85万円～760万円）の範囲で、平均値は22,640ドル（約34万円）です。
75%の値が25,22ドルですから、高価格の住宅は少ないことがわかります。

グラフでデータを分析する

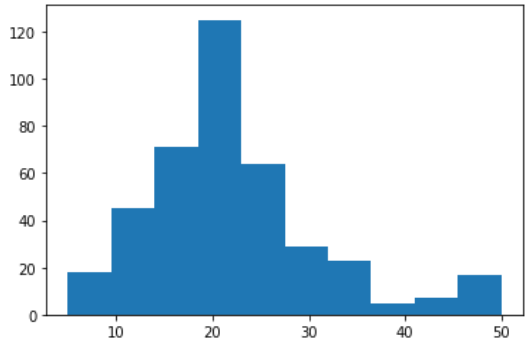
様々なグラフで、データを分析してみましょう。

ヒストグラムで値の分布を確認する

値の分布は数字だけでは分かりづらいので、住宅価格をヒストグラムで描画してみましょう。
ヒストグラムは `plt.hist()` で描画することができます。

```
plt.hist(y_train)
```

```
(array([ 18.,  45.,  71., 125.,  64.,  29.,  23.,   5.,   7.,  17.]),
 array([ 5. ,  9.5, 14. , 18.5, 23. , 27.5, 32. , 36.5, 41. , 45.5, 50. ]),
 <a list of 10 Patch objects>)
```



x軸が住宅価格、y軸はデータの件数です。
20,000ドルの周辺が最もデータ件数が多く、40,000ドル周辺の価格帯のデータが少ないことがわかります。

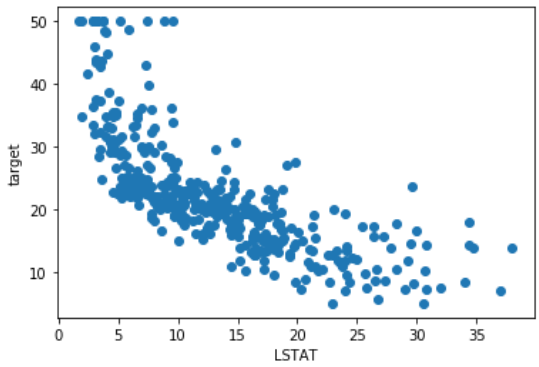
散布図で分布と相関を確認する

x_train には13種類の値が含まれていました。
それぞれの値は、住宅価格にどの程度影響しているのでしょうか。

ディープラーニングは特徴量を自動で見つけ出してくれる手法ですが、項目が多ければ学習に時間がかかります。
また、全く相関のない項目は時にノイズになってしまいますので、あらかじめ除外しましょう。

散布図を描画して確認してみましょう。
散布図は `plt.scatter()` で描画できます。

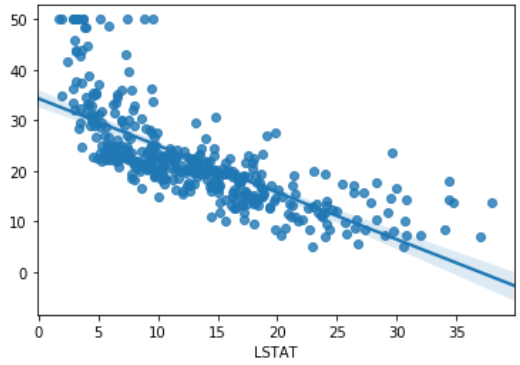
```
# 低所得者の割合と住宅価格の散布図
plt.figure()
plt.scatter(x=x_train_df['LSTAT'], y=y_train)
plt.xlabel('LSTAT')
plt.ylabel('target')
plt.show()
```



上記のグラフはseabornでも描画できます。

```
# 低所得者の割合と住宅価格の散布図（回帰直線つき）
sns.regplot(x=x_train_df['LSTAT'], y=y_train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0f5dc630f0>
```

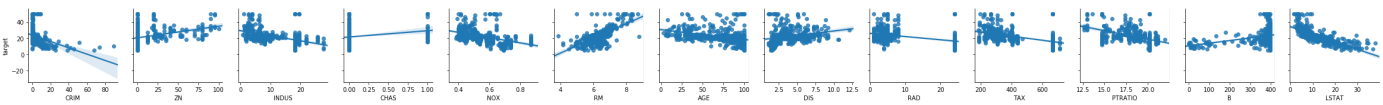


低所得者の割合と住宅価格には、負の相関があることがわかります。
低所得者の割合は、住宅価格を決める要因となっていると言えるでしょう。

住宅価格と他の項目の散布図を一気に描画するには、`sns.pairplot()` を使います。

```
# x_train_dfとy_train_dfを一時的にまとめる
df = x_train_df.copy()
df['target'] = y_train_df

# 散布図（回帰直線つき）を出力
sns.pairplot(data=df, x_vars=columns_name, y_vars='target', kind='reg')
```

```
<seaborn.axisgrid.PairGrid at 0x7f0f5aec6198>
```

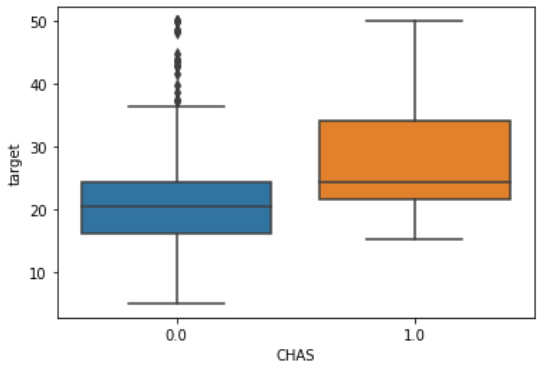


箱ひげ図で相関を確認する

CHASはカテゴリ変数（連続値ではなく、人が意味を決めた値）ですので、散布図ではよくわかりません。
箱ひげ図を描画してみましょう。

```
# 箱ひげ図の出力
sns.boxplot(data=df, x='CHAS', y='target')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0f5aa9a128>
```



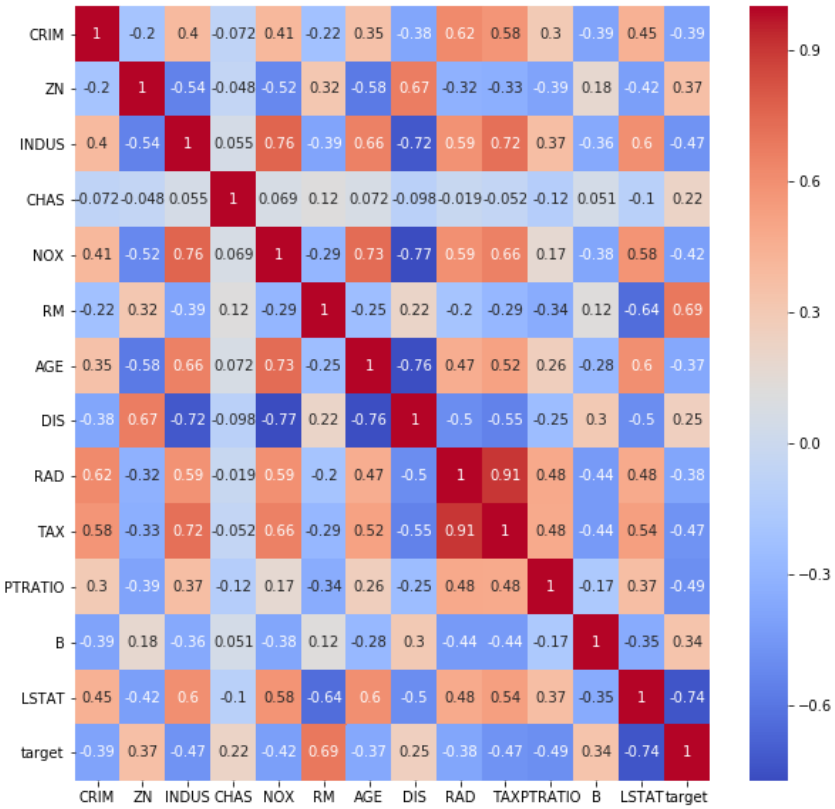
0（川に面していない）よりも、1（川に面している）方が、住宅価格が高い傾向にあることがわかりました。
住宅価格を決める要因になっていそうです。

ヒートマップで相関を確認する

特徴量間の相関をヒートマップで描画してみましょう。

```
# 相関係数を計算
df_corr = df.corr()

# ヒートマップを描画
plt.figure(figsize=(10, 10))
sns.heatmap(df_corr, annot=True, cmap='coolwarm')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0f591c9748>
```



RADとTAXは高い相関があることがわかりました。
このような高い相関値をもつ説明変数を複数使うと、信頼性が低下する「多重共線性（multicollinearity、マルチコ）」と呼ばれる減少が発生しますので、どちらか一方を除外します。

データの前処理

学習の前にデータを前処理しましょう。

特徴量を選択する

分析の結果、RAD と TAX は高い相関がありました。
多重共線性を防ぐため、 RAD は除外することにします。

```
# RAD列を削除
x_train_df = x_train_df.drop('RAD', axis=1)
x_test_df = x_test_df.drop('RAD', axis=1)
```

訓練データ、検証データ、テストデータに分割する

このデータセットは読み込んだ時点で訓練データとテストデータに分かれていました。
もし自作のデータセットなどを使う場合は、ここでデータを分割します。

- 訓練データ … 学習に使用する教師データ
- 検証データ … 学習の過程で精度を判断するためのデータ
- テストデータ … 作成したモデルの精度を評価するためのデータ

訓練データの1割を検証データとして分割しておきましょう。

```
# 訓練データのうち1割を検証データにする
(x_train_df, x_val_df, y_train_df, y_val_df) = train_test_split(x_train_df, y_train_df, test_size=0.1, random_state=777)

print('訓練データの件数', len(x_train_df))
print('検証データの件数', len(x_val_df))
```

```
訓練データの件数 363
検証データの件数 41
```

標準化

データセットに含まれる値は、特徴量の種類によって値の範囲がバラバラです。
特徴量の範囲が広いまま学習を行うと、収束に時間がかかるばかりか、最悪いつまでたっても収束しないという事態が起こります。
そこで、各特徴量の平均が0、標準偏差が1になるよう整形します。
この作業を **標準化** といいます。

標準化するには、各特徴量について、平均値を引いて標準偏差で割ります。

```
# 各列の平均
mean = x_train_df.mean()
# 各列の標準偏差
std = x_train_df.std(ddof=False)

# 平均を引き、標準偏差で割る
x_train_df -= mean
x_train_df /= std

x_train_df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	TAX	PTRATIO	B	LSTAT
209	-0.28	-0.47	1.14	-0.29	0.36	-0.28	1.06	-0.94	-0.07	-1.69	-0.02	-0.90
110	-0.03	-0.47	0.93	-0.29	1.30	-1.84	0.75	-0.94	1.47	0.81	-0.43	0.12
239	0.70	-0.47	0.93	-0.29	1.48	0.28	0.87	-0.83	1.47	0.81	-3.38	1.47
161	0.47	-0.47	0.93	-0.29	0.19	0.12	0.56	-0.80	1.47	0.81	-2.94	0.62
147	-0.41	-0.47	-0.44	-0.29	-0.34	-0.09	-0.58	-0.46	-0.18	1.13	0.41	-0.01

```
# 基本統計
x_train_df.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	TAX	PTRATIO	B	LSTAT

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	TAX	PTRATIO	B	LSTAT
count	363.00	363.00	363.00	363.00	363.00	363.00	363.00	363.00	363.00	363.00	363.00	363.00
mean	-0.00	0.00	0.00	-0.00	-0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
std	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
min	-0.42	-0.47	-1.56	-0.29	-1.46	-3.40	-2.43	-1.23	-1.33	-2.65	-3.84	-1.54
25%	-0.41	-0.47	-0.88	-0.29	-0.92	-0.55	-0.84	-0.80	-0.79	-0.65	0.21	-0.77
50%	-0.39	-0.47	-0.23	-0.29	-0.19	-0.08	0.40	-0.31	-0.46	0.26	0.39	-0.15
75%	0.01	-0.21	0.93	-0.29	0.71	0.50	0.86	0.67	1.47	0.81	0.44	0.55
max	8.87	3.76	2.31	3.46	2.57	3.57	1.06	3.95	1.73	1.63	0.45	3.37

平均（mean）が 0 、分散（std）が 1 になりました。

検証データとテストデータにも同様の処理を行います。
この時使う平均値・標準偏差値は、元の訓練データのものです。
学習に使用した基準と同じ基準で整形しておかなくては、正しい結果が出ないからです。

```
# 検証データ、テストデータも同様に処理
x_val_df -= mean
x_val_df /= std

x_test_df -= mean
x_test_df /= std
```

モデルの定義

モデルの形を決める

今回は訓練データが363件と少ないので、浅いニューラルネットワークモデルにします。
データ数が少ないのに深いニューラルネットワークモデルにしてしまうと、過学習が発生しやすくなるからです。

`Sequential` モデルは、層（レイヤー）を線形に積み重ねる基本的な形です。
`layers.Dense` は結合層です。

層（レイヤー）の指定

入力の形

最初の層の `input_shape` には、入力の形状、今回は特徴量の数（列数）を渡す必要があります。
2層目以降は、前の層の出力を自動で受け取りますので、指定する必要はありません。

出力ユニット数

隠れ層には、任意の出力ユニット数（2の累乗数）を指定します。
特徴量の情報は、その層で出力ユニット数に合わせて抽象化されます。
ユニット数は小さすぎても、大きすぎてもいけません。
小さいと汎化性能は上がりますが、学習に時間がかかったり、汎化されすぎて性能が下がってしまいます。
逆に大きいと学習は速く進みますが、訓練データに適合しすぎてしまい過学習が発生します。
特徴量の数やデータ数に合わせて選択しましょう。

活性化関数

活性化関数はReLuを指定します。現在最も使われている活性化関数です。

出力層の出力ユニット数、活性化関数

今回は連続値を1つだけ出力したい（スカラー回帰）ので、最後の出力層は出力ユニット数を 1 にします。
また、出力された値をそのまま使用しますので、活性化関数は使用しません。

```
# モデルの定義
model = tf.keras.Sequential([

    # --- 隠れ層 --- #
```

```
    layers.Dense(128, activation='relu', input_shape=[len(x_train_df.columns)]),  # 最初の層はinput_shapeを指定する
    layers.Dense(128, activation='relu'),

    # --- 出力層 --- #

    layers.Dense(1)  # 出力層のユニット数は1
])
```

定義したモデルの情報を見てみましょう。

```
# モデルの構造情報
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               1664
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 18,305
Trainable params: 18,305
Non-trainable params: 0
_________________________________________________________________
```

トータルのパラメータ数は18,305です。
パラメータ数は出力ユニット数や説明変数の数によって変わります。
これだけの数のパラメータが、学習によって自動的に調整されていきます。

モデルのコンパイル

最適化アルゴリズム、損失関数、評価関数を指定します。

- 最適化アルゴリズム … 学習の進み方を決めるアルゴリズムです。RMSProp、Adamがよく使われます。
- 損失関数 … 学習は損失関数の出力値が小さくなる方向に進められます。回帰問題では、平均二乗誤差（MSE:mean_squared_error）がよく使われます。
- 評価関数 … 人間が精度を評価するために使用します。学習には影響しません。回帰問題では、予測値が正解からどれだけ離れているかを調べるために、平均絶対誤差（MAE:mean_absolute_error）を使うことが多いです。

```
# モデルのコンパイル
model.compile(optimizer='adam',                     # 最適化アルゴリズム…Adam
              loss='mean_squared_error',            # 損失関数…MSE（平均二乗誤差）
              metrics=['mean_absolute_error'])      # 評価関数…MAE（平均絶対誤差）
```

学習

いよいよ、学習を行いましょう。
学習は `model.fit()` で行います。

ハイパーパラメータ

人間があらかじめ指定する値を、ハイパーパラメータといいます。
今回の学習で指定するバッチサイズ `batch_size` とエポック `epoches` はハイパーパラメータです。
最初はどの値がいいか分からないので、適当な値で試します。

バッチサイズ batch_size

バッチサイズは、データをサブセットに分割して学習する際、何件ずつのサブセットにするかの指定です。
小さいと１つ１つのデータに敏感に反応します。学習に時間がかかります。
大きいとデータの影響は平均化され、学習は速く終わります。

エポック spoches

エポックは、何回学習を繰り返すかの指定です。
少ないと学習は速く終わりますが、十分には学習されません。
大きすぎると過学習が発生します。

検証データ validation_data

エポック終了ごとに、検証データを使ってモデルを評価します。

```
# 学習とエポックごとの評価
history = model.fit(x_train_df,                          # 訓練データ
                    y_train_df,                          # 訓練データのラベル
                    batch_size=32,                       # バッチサイズ
                    epochs=20,                           # エポック数
                    validation_data=(x_val_df, y_val_df))  # 検証データ
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class
'pandas.core.frame.DataFrame'>, <class 'NoneType'>
Train on 363 samples, validate on 41 samples
Epoch 1/20
363/363 [==============================] - 0s 909us/sample - loss: 523.0075 - mean_absolute_error: 21.0983 - val_loss:
587.4126 - val_mean_absolute_error: 21.7300
Epoch 2/20
363/363 [==============================] - 0s 128us/sample - loss: 423.7093 - mean_absolute_error: 18.7258 - val_loss:
453.2799 - val_mean_absolute_error: 18.6719
Epoch 3/20
363/363 [==============================] - 0s 119us/sample - loss: 286.8546 - mean_absolute_error: 14.9921 - val_loss:
271.6295 - val_mean_absolute_error: 13.6792
Epoch 4/20
363/363 [==============================] - 0s 115us/sample - loss: 143.4867 - mean_absolute_error: 9.9356 - val_loss:
120.7345 - val_mean_absolute_error: 7.8364
Epoch 5/20
363/363 [==============================] - 0s 126us/sample - loss: 66.7433 - mean_absolute_error: 6.4603 - val_loss:
76.2880 - val_mean_absolute_error: 6.5867
Epoch 6/20
363/363 [==============================] - 0s 124us/sample - loss: 48.7322 - mean_absolute_error: 5.3565 - val_loss:
57.6279 - val_mean_absolute_error: 5.5677
Epoch 7/20
363/363 [==============================] - 0s 114us/sample - loss: 33.7071 - mean_absolute_error: 4.2994 - val_loss:
51.6475 - val_mean_absolute_error: 5.0210
Epoch 8/20
363/363 [==============================] - 0s 108us/sample - loss: 27.0545 - mean_absolute_error: 3.7098 - val_loss:
45.1796 - val_mean_absolute_error: 4.6186
Epoch 9/20
363/363 [==============================] - 0s 124us/sample - loss: 23.7617 - mean_absolute_error: 3.4146 - val_loss:
40.9632 - val_mean_absolute_error: 4.3930
Epoch 10/20
363/363 [==============================] - 0s 130us/sample - loss: 21.7450 - mean_absolute_error: 3.2368 - val_loss:
41.0543 - val_mean_absolute_error: 4.2489
Epoch 11/20
363/363 [==============================] - 0s 122us/sample - loss: 20.6567 - mean_absolute_error: 3.1117 - val_loss:
38.7872 - val_mean_absolute_error: 4.1730
Epoch 12/20
363/363 [==============================] - 0s 132us/sample - loss: 19.6076 - mean_absolute_error: 3.0015 - val_loss:
39.0042 - val_mean_absolute_error: 4.0891
Epoch 13/20
363/363 [==============================] - 0s 132us/sample - loss: 18.3732 - mean_absolute_error: 2.9112 - val_loss:
35.9138 - val_mean_absolute_error: 3.9978
Epoch 14/20
363/363 [==============================] - 0s 116us/sample - loss: 17.9652 - mean_absolute_error: 2.8993 - val_loss:
33.3214 - val_mean_absolute_error: 3.9265
Epoch 15/20
363/363 [==============================] - 0s 111us/sample - loss: 17.1291 - mean_absolute_error: 2.8220 - val_loss:
33.9916 - val_mean_absolute_error: 3.9178
Epoch 16/20
363/363 [==============================] - 0s 125us/sample - loss: 16.6976 - mean_absolute_error: 2.7472 - val_loss:
34.9649 - val_mean_absolute_error: 3.8571
Epoch 17/20
363/363 [==============================] - 0s 121us/sample - loss: 15.9733 - mean_absolute_error: 2.7044 - val_loss:
32.1469 - val_mean_absolute_error: 3.7642
```

```
Epoch 18/20
363/363 [==============================] - 0s 129us/sample - loss: 15.5895 - mean_absolute_error: 2.6688 - val_loss:
32.4352 - val_mean_absolute_error: 3.7636
Epoch 19/20
363/363 [==============================] - 0s 121us/sample - loss: 15.1298 - mean_absolute_error: 2.6122 - val_loss:
32.8204 - val_mean_absolute_error: 3.7550
Epoch 20/20
363/363 [==============================] - 0s 122us/sample - loss: 14.9545 - mean_absolute_error: 2.5747 - val_loss:
32.6500 - val_mean_absolute_error: 3.6947
```

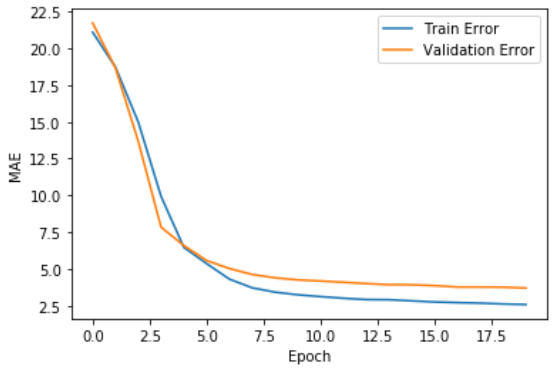
エポックごとの出力を詳しく見てみましょう。

項目	内容
loss	訓練データに対する損失関数（MSE）の値
mean_absolute_error	訓練データに対する評価関数（MAE）の値
val_loss	検証データに対する損失関数（MSE）の値
val_mean_absolute_error	検証データに対する評価関数（MAE）の値

学習ではlossが小さくなるようにパラメータが調整されます。
エポックを繰り返すごとに、loss が小さくなっていることがわかります。

学習の推移をグラフで描画してみましょう。

```
plt.figure()
plt.plot(history.epoch, history.history['mean_absolute_error'], label='Train Error')
plt.plot(history.epoch, history.history['val_mean_absolute_error'], label='Validation Error')
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.legend()
plt.show()
```



評価

テストデータを使って、モデルの性能を評価しましょう。
モデルの評価は `evaluate()` で行います。
モデルのコンパイル時に指定した、損失関数・評価関数の値が出力されます。

```
# 評価データを使って性能を調べる
loss, mae = model.evaluate(x_test_df, y_test_df)
print('損失:', loss, 'MAE:', mae)
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class
'pandas.core.frame.DataFrame'>, <class 'NoneType'>
102/102 [==============================] - 0s 160us/sample - loss: 12.5922 - mean_absolute_error: 2.6206
損失: 12.592161758273255 MAE: 2.6206172
```

モデルの保存

学習済みのモデルを保存し、いつでも使えるようにしましょう。
`model.save()` で保存します。拡張子は「h5」です。
モデルの構造や、調整されたパラメータも保存されます。

GoogleColabratory で実行している場合は、実行環境に保存しても、セッションが切れるとデータが消去されてしまいます。
そこで、Googleドライブをマウントして、Googleドライブに保存します。

```
# Google ドライブをマウント
# 表示されるURLにアクセスしてコードを取得し、下の入力欄に入力
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive",
force_remount=True).
```

```
# 保存パス
save_path = '/content/drive/My Drive/my_boston_model.h5'

# モデルの保存
model.save(save_path)
```

保存してあるモデルは、`tf.keras.models.load_model()` を使って読み込みます。
学習済みのモデルは推論に使うことも、再学習することもできます。

```
# 保存されているモデルの読込
model_loaded = models.load_model(save_path)

# 評価
loss, mae = model_loaded.evaluate(x_test_df, y_test_df)
print('損失:', loss, 'MAE:', mae)
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class
'pandas.core.frame.DataFrame'>, <class 'NoneType'>
102/102 [==============================] - 0s 270us/sample - loss: 12.5922 - mean_absolute_error: 2.6206
損失: 12.592161758273255 MAE: 2.6206172
```

推論

作成したモデルで、住宅価格を予測しましょう。
推論は `predict()` で行います。

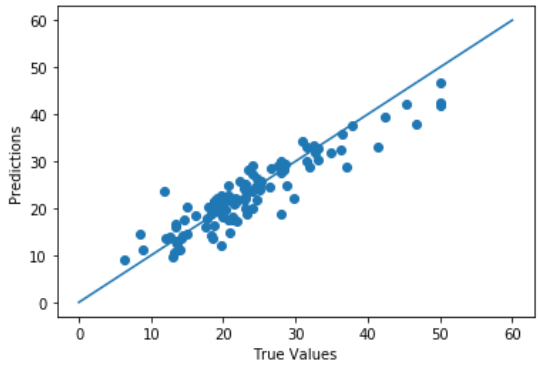
```
# 推論
y_pred = model.predict(x_test_df)

print('予測値 | 実際の値')
for i in range(10):
  print(f'{y_pred[i, 0]:.2f} | {y_test[i]:.1f}')
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class
'pandas.core.frame.DataFrame'>, <class 'NoneType'>
予測値 | 実際の値
20.86 | 21.4
19.90 | 24.0
42.46 | 50.0
23.67 | 24.1
12.32 | 13.5
37.64 | 37.9
28.28 | 28.4
28.91 | 37.0
```

```
28.63 | 26.6
24.52 | 24.4
```

予測値と正解にどれだけ差があったか、グラフで描画してみましょう。

```
# 実際の値と予測した値をグラフで表示
plt.scatter(x=y_test, y=y_pred)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.plot([0, 60], [0, 60])  # 正解の直線
plt.show()
```



チューニング

作成したモデルはあまり精度がよくありませんでしたので、どうやったら精度が改善するかを考えましょう。
一番効果があるのは、**データ数を増やすこと**です。
しかし今回はデータ数が限られていますので、他の方法で精度向上を目指します。

- モデルの改良
    - 隠れ層の出力ユニット数を変える
    - 隠れ層の数を変える
    - 最適化関数を変える
    - ドロップアウト層を追加する（過学習対策）
    - 正則化を行う（過学習対策）
- ハイパーパラメータの改良
    - バッチサイズを変える
    - エポック数を変える
    - 最適化関数の学習率を変える

訓練データの損失値が十分に下がりきっていない場合は、エポック数を増やしてみます。
学習の推移のグラフの曲線が下がっていない場合、出力ユニット数や隠れ層の数、バッチサイズが小さすぎたり、大きすぎていないか確認しましょう。

訓練データの損失値は小さいのに、検証データの損失値が小さくならない場合は過学習に陥っています。ドロップアウト層を追加したり、正則化を行います。

チューニングによって精度がどう変わるか、色々と試してみて下さい。

```
# モデルの定義
model = tf.keras.Sequential([

    # --- 隠れ層 --- #

    layers.Dense(128, activation='relu', input_shape=[len(x_train_df.columns)]),
    layers.Dense(128, activation='relu'),

    layers.Dropout(0.2),  # ドロップアウト層

    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),

    layers.Dropout(0.2),
```

```
    layers.Dense(32, activation='relu'),

    layers.Dropout(0.1),

    # --- 出力層 --- #
    layers.Dense(1)

])

# モデルのコンパイル
model.compile(optimizer='adam',                      # 最適化アルゴリズム…Adam
              loss='mean_squared_error',             # 損失関数…MSE（平均二乗誤差）
              metrics=['mean_absolute_error'])       # 評価関数…MAE（平均絶対誤差）

# 学習とエポックごとの評価
history = model.fit(x_train_df,                          # 訓練データ
                    y_train_df,                          # 訓練データのラベル
                    batch_size=32,                       # バッチサイズ
                    epochs=100,                          # エポック数
                    validation_data=(x_val_df, y_val_df))  # 検証データ

# 学習の推移をグラフ描画
plt.figure()
plt.plot(history.epoch, history.history['mean_absolute_error'], label='Train Error')
plt.plot(history.epoch, history.history['val_mean_absolute_error'], label='Validation Error')
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.legend()
plt.show()

# 評価データを使って性能を調べる
loss, mae = model.evaluate(x_test_df, y_test_df)
print('損失:', loss, 'MAE:', mae)
```

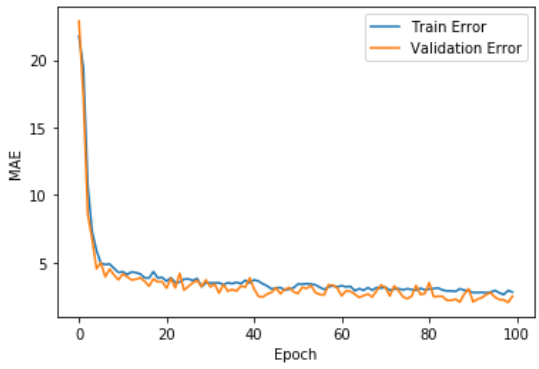
```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class
'pandas.core.frame.DataFrame'>, <class 'NoneType'>
Train on 363 samples, validate on 41 samples
Epoch 1/100
363/363 [==============================] - 0s 897us/sample - loss: 554.4624 - mean_absolute_error: 21.7639 - val_loss:
645.9646 - val_mean_absolute_error: 22.8784
Epoch 2/100
363/363 [==============================] - 0s 146us/sample - loss: 460.0329 - mean_absolute_error: 19.4270 - val_loss:
416.3242 - val_mean_absolute_error: 17.3020
Epoch 3/100
363/363 [==============================] - 0s 155us/sample - loss: 187.9852 - mean_absolute_error: 10.7484 - val_loss:
109.4609 - val_mean_absolute_error: 8.5874
Epoch 4/100
363/363 [==============================] - 0s 167us/sample - loss: 97.1083 - mean_absolute_error: 7.3128 - val_loss:
94.1733 - val_mean_absolute_error: 6.7504
Epoch 5/100
363/363 [==============================] - 0s 154us/sample - loss: 62.4467 - mean_absolute_error: 5.9042 - val_loss:
39.7728 - val_mean_absolute_error: 4.5554
Epoch 6/100
363/363 [==============================] - 0s 160us/sample - loss: 41.9286 - mean_absolute_error: 4.9534 - val_loss:
47.5429 - val_mean_absolute_error: 5.0028
Epoch 7/100
363/363 [==============================] - 0s 167us/sample - loss: 44.4635 - mean_absolute_error: 4.8605 - val_loss:
33.2480 - val_mean_absolute_error: 3.9582
Epoch 8/100
363/363 [==============================] - 0s 158us/sample - loss: 44.7637 - mean_absolute_error: 4.9042 - val_loss:
45.4019 - val_mean_absolute_error: 4.5287
Epoch 9/100
363/363 [==============================] - 0s 168us/sample - loss: 38.0730 - mean_absolute_error: 4.5845 - val_loss:
37.1055 - val_mean_absolute_error: 4.0895
Epoch 10/100
363/363 [==============================] - 0s 165us/sample - loss: 33.9760 - mean_absolute_error: 4.2766 - val_loss:
31.0214 - val_mean_absolute_error: 3.7362
Epoch 11/100
363/363 [==============================] - 0s 155us/sample - loss: 35.3772 - mean_absolute_error: 4.3325 - val_loss:
37.5383 - val_mean_absolute_error: 4.1608
Epoch 12/100
```

```
363/363 [==============================] - 0s 155us/sample - loss: 29.9424 - mean_absolute_error: 4.1333 - val_loss:
33.1990 - val_mean_absolute_error: 3.9618
Epoch 13/100
363/363 [==============================] - 0s 147us/sample - loss: 31.8019 - mean_absolute_error: 4.3156 - val_loss:
30.5598 - val_mean_absolute_error: 3.7277
Epoch 14/100
363/363 [==============================] - 0s 148us/sample - loss: 33.9637 - mean_absolute_error: 4.2683 - val_loss:
31.4661 - val_mean_absolute_error: 3.7767
Epoch 15/100
363/363 [==============================] - 0s 154us/sample - loss: 31.0798 - mean_absolute_error: 4.1648 - val_loss:
33.0022 - val_mean_absolute_error: 3.8802
Epoch 16/100
363/363 [==============================] - 0s 149us/sample - loss: 28.1683 - mean_absolute_error: 3.8848 - val_loss:
29.5154 - val_mean_absolute_error: 3.6273
Epoch 17/100
363/363 [==============================] - 0s 142us/sample - loss: 28.1501 - mean_absolute_error: 3.8608 - val_loss:
26.0716 - val_mean_absolute_error: 3.2605
Epoch 18/100
363/363 [==============================] - 0s 161us/sample - loss: 31.9209 - mean_absolute_error: 4.3372 - val_loss:
30.6887 - val_mean_absolute_error: 3.8004
Epoch 19/100
363/363 [==============================] - 0s 144us/sample - loss: 27.7185 - mean_absolute_error: 3.8726 - val_loss:
27.3077 - val_mean_absolute_error: 3.5798
Epoch 20/100
363/363 [==============================] - 0s 191us/sample - loss: 26.3359 - mean_absolute_error: 3.9127 - val_loss:
29.3432 - val_mean_absolute_error: 3.6011
Epoch 21/100
363/363 [==============================] - 0s 157us/sample - loss: 23.1904 - mean_absolute_error: 3.6320 - val_loss:
23.0371 - val_mean_absolute_error: 3.1176
Epoch 22/100
363/363 [==============================] - 0s 168us/sample - loss: 27.7543 - mean_absolute_error: 3.8853 - val_loss:
31.0766 - val_mean_absolute_error: 3.8307
Epoch 23/100
363/363 [==============================] - 0s 152us/sample - loss: 22.6743 - mean_absolute_error: 3.5249 - val_loss:
23.1177 - val_mean_absolute_error: 3.1404
Epoch 24/100
363/363 [==============================] - 0s 145us/sample - loss: 22.2700 - mean_absolute_error: 3.5635 - val_loss:
33.8279 - val_mean_absolute_error: 4.2106
Epoch 25/100
363/363 [==============================] - 0s 147us/sample - loss: 24.6719 - mean_absolute_error: 3.7846 - val_loss:
21.0389 - val_mean_absolute_error: 2.9763
Epoch 26/100
363/363 [==============================] - 0s 158us/sample - loss: 27.0022 - mean_absolute_error: 3.7958 - val_loss:
22.9295 - val_mean_absolute_error: 3.2442
Epoch 27/100
363/363 [==============================] - 0s 163us/sample - loss: 23.8926 - mean_absolute_error: 3.6903 - val_loss:
28.6231 - val_mean_absolute_error: 3.5216
Epoch 28/100
363/363 [==============================] - 0s 173us/sample - loss: 26.5721 - mean_absolute_error: 3.8299 - val_loss:
30.4897 - val_mean_absolute_error: 3.6863
Epoch 29/100
363/363 [==============================] - 0s 148us/sample - loss: 18.8830 - mean_absolute_error: 3.2204 - val_loss:
23.9200 - val_mean_absolute_error: 3.2386
Epoch 30/100
363/363 [==============================] - 0s 165us/sample - loss: 21.8840 - mean_absolute_error: 3.5341 - val_loss:
28.5816 - val_mean_absolute_error: 3.7306
Epoch 31/100
363/363 [==============================] - 0s 148us/sample - loss: 21.5996 - mean_absolute_error: 3.4988 - val_loss:
23.8071 - val_mean_absolute_error: 3.2158
Epoch 32/100
363/363 [==============================] - 0s 154us/sample - loss: 21.6129 - mean_absolute_error: 3.5043 - val_loss:
25.0016 - val_mean_absolute_error: 3.4126
Epoch 33/100
363/363 [==============================] - 0s 158us/sample - loss: 22.7584 - mean_absolute_error: 3.5153 - val_loss:
17.0237 - val_mean_absolute_error: 2.7546
Epoch 34/100
363/363 [==============================] - 0s 150us/sample - loss: 20.1522 - mean_absolute_error: 3.3654 - val_loss:
24.4904 - val_mean_absolute_error: 3.3898
Epoch 35/100
363/363 [==============================] - 0s 164us/sample - loss: 20.6818 - mean_absolute_error: 3.5177 - val_loss:
18.1097 - val_mean_absolute_error: 2.8959
Epoch 36/100
```

```
363/363 [==============================] - 0s 159us/sample - loss: 21.1354 - mean_absolute_error: 3.4512 - val_loss:
18.7466 - val_mean_absolute_error: 3.0158
Epoch 37/100
363/363 [==============================] - 0s 176us/sample - loss: 21.9615 - mean_absolute_error: 3.5492 - val_loss:
19.4462 - val_mean_absolute_error: 2.9058
Epoch 38/100
363/363 [==============================] - 0s 148us/sample - loss: 20.4335 - mean_absolute_error: 3.4523 - val_loss:
22.4754 - val_mean_absolute_error: 3.2851
Epoch 39/100
363/363 [==============================] - 0s 171us/sample - loss: 23.7917 - mean_absolute_error: 3.7000 - val_loss:
22.4718 - val_mean_absolute_error: 3.1849
Epoch 40/100
363/363 [==============================] - 0s 155us/sample - loss: 20.9181 - mean_absolute_error: 3.5084 - val_loss:
29.6153 - val_mean_absolute_error: 3.8717
Epoch 41/100
363/363 [==============================] - 0s 155us/sample - loss: 24.7226 - mean_absolute_error: 3.7328 - val_loss:
21.0243 - val_mean_absolute_error: 3.0557
Epoch 42/100
363/363 [==============================] - 0s 150us/sample - loss: 21.3511 - mean_absolute_error: 3.6369 - val_loss:
13.7140 - val_mean_absolute_error: 2.5087
Epoch 43/100
363/363 [==============================] - 0s 145us/sample - loss: 19.6900 - mean_absolute_error: 3.4333 - val_loss:
14.0817 - val_mean_absolute_error: 2.4661
Epoch 44/100
363/363 [==============================] - 0s 157us/sample - loss: 20.0514 - mean_absolute_error: 3.2763 - val_loss:
16.1347 - val_mean_absolute_error: 2.6890
Epoch 45/100
363/363 [==============================] - 0s 169us/sample - loss: 17.2421 - mean_absolute_error: 3.0465 - val_loss:
18.9695 - val_mean_absolute_error: 2.8071
Epoch 46/100
363/363 [==============================] - 0s 185us/sample - loss: 18.5656 - mean_absolute_error: 3.1390 - val_loss:
20.5616 - val_mean_absolute_error: 3.1452
Epoch 47/100
363/363 [==============================] - 0s 160us/sample - loss: 17.2207 - mean_absolute_error: 3.1536 - val_loss:
16.9178 - val_mean_absolute_error: 2.6949
Epoch 48/100
363/363 [==============================] - 0s 163us/sample - loss: 15.1238 - mean_absolute_error: 2.9882 - val_loss:
20.9629 - val_mean_absolute_error: 3.0263
Epoch 49/100
363/363 [==============================] - 0s 159us/sample - loss: 15.5593 - mean_absolute_error: 3.0378 - val_loss:
19.4941 - val_mean_absolute_error: 3.1501
Epoch 50/100
363/363 [==============================] - 0s 154us/sample - loss: 17.9596 - mean_absolute_error: 3.1295 - val_loss:
17.8021 - val_mean_absolute_error: 2.8531
Epoch 51/100
363/363 [==============================] - 0s 156us/sample - loss: 20.1790 - mean_absolute_error: 3.4168 - val_loss:
16.9996 - val_mean_absolute_error: 2.7237
Epoch 52/100
363/363 [==============================] - 0s 153us/sample - loss: 20.0266 - mean_absolute_error: 3.4132 - val_loss:
15.2833 - val_mean_absolute_error: 3.1808
Epoch 53/100
363/363 [==============================] - 0s 171us/sample - loss: 20.3678 - mean_absolute_error: 3.4460 - val_loss:
19.0247 - val_mean_absolute_error: 3.0889
Epoch 54/100
363/363 [==============================] - 0s 156us/sample - loss: 21.3393 - mean_absolute_error: 3.4151 - val_loss:
21.3343 - val_mean_absolute_error: 3.3160
Epoch 55/100
363/363 [==============================] - 0s 162us/sample - loss: 19.6874 - mean_absolute_error: 3.3635 - val_loss:
14.7127 - val_mean_absolute_error: 2.7891
Epoch 56/100
363/363 [==============================] - 0s 152us/sample - loss: 17.8876 - mean_absolute_error: 3.1771 - val_loss:
15.1379 - val_mean_absolute_error: 2.6477
Epoch 57/100
363/363 [==============================] - 0s 166us/sample - loss: 16.0852 - mean_absolute_error: 3.0170 - val_loss:
13.9153 - val_mean_absolute_error: 2.6128
Epoch 58/100
363/363 [==============================] - 0s 151us/sample - loss: 16.4935 - mean_absolute_error: 3.2127 - val_loss:
20.7351 - val_mean_absolute_error: 3.3635
Epoch 59/100
363/363 [==============================] - 0s 147us/sample - loss: 17.9395 - mean_absolute_error: 3.2889 - val_loss:
22.4995 - val_mean_absolute_error: 3.3315
Epoch 60/100
```

```
363/363 [==============================] - 0s 156us/sample - loss: 19.0806 - mean_absolute_error: 3.2175 - val_loss:
19.7369 - val_mean_absolute_error: 3.1258
Epoch 61/100
363/363 [==============================] - 0s 153us/sample - loss: 19.1782 - mean_absolute_error: 3.2950 - val_loss:
12.0653 - val_mean_absolute_error: 2.5530
Epoch 62/100
363/363 [==============================] - 0s 161us/sample - loss: 18.4768 - mean_absolute_error: 3.2133 - val_loss:
17.5311 - val_mean_absolute_error: 2.9174
Epoch 63/100
363/363 [==============================] - 0s 156us/sample - loss: 19.4713 - mean_absolute_error: 3.2485 - val_loss:
16.6706 - val_mean_absolute_error: 2.8780
Epoch 64/100
363/363 [==============================] - 0s 152us/sample - loss: 15.2336 - mean_absolute_error: 2.9213 - val_loss:
15.4764 - val_mean_absolute_error: 2.6669
Epoch 65/100
363/363 [==============================] - 0s 149us/sample - loss: 17.0277 - mean_absolute_error: 3.0796 - val_loss:
12.3316 - val_mean_absolute_error: 2.4288
Epoch 66/100
363/363 [==============================] - 0s 142us/sample - loss: 15.0951 - mean_absolute_error: 2.9592 - val_loss:
13.9010 - val_mean_absolute_error: 2.5682
Epoch 67/100
363/363 [==============================] - 0s 159us/sample - loss: 18.1342 - mean_absolute_error: 3.1474 - val_loss:
14.1946 - val_mean_absolute_error: 2.6753
Epoch 68/100
363/363 [==============================] - 0s 170us/sample - loss: 16.0173 - mean_absolute_error: 2.9684 - val_loss:
14.9348 - val_mean_absolute_error: 2.4630
Epoch 69/100
363/363 [==============================] - 0s 145us/sample - loss: 16.7295 - mean_absolute_error: 3.1614 - val_loss:
17.5353 - val_mean_absolute_error: 2.9316
Epoch 70/100
363/363 [==============================] - 0s 181us/sample - loss: 16.4868 - mean_absolute_error: 3.1141 - val_loss:
20.6480 - val_mean_absolute_error: 3.3460
Epoch 71/100
363/363 [==============================] - 0s 158us/sample - loss: 17.3486 - mean_absolute_error: 3.1799 - val_loss:
21.1057 - val_mean_absolute_error: 3.1538
Epoch 72/100
363/363 [==============================] - 0s 143us/sample - loss: 14.9060 - mean_absolute_error: 2.9567 - val_loss:
12.5331 - val_mean_absolute_error: 2.5365
Epoch 73/100
363/363 [==============================] - 0s 169us/sample - loss: 16.5181 - mean_absolute_error: 3.0479 - val_loss:
18.8989 - val_mean_absolute_error: 3.2770
Epoch 74/100
363/363 [==============================] - 0s 146us/sample - loss: 16.1044 - mean_absolute_error: 3.0769 - val_loss:
16.8598 - val_mean_absolute_error: 2.9243
Epoch 75/100
363/363 [==============================] - 0s 153us/sample - loss: 16.2694 - mean_absolute_error: 2.9939 - val_loss:
14.8342 - val_mean_absolute_error: 2.4845
Epoch 76/100
363/363 [==============================] - 0s 156us/sample - loss: 15.9368 - mean_absolute_error: 3.0700 - val_loss:
11.7365 - val_mean_absolute_error: 2.3332
Epoch 77/100
363/363 [==============================] - 0s 144us/sample - loss: 15.1171 - mean_absolute_error: 3.0131 - val_loss:
15.8869 - val_mean_absolute_error: 2.5266
Epoch 78/100
363/363 [==============================] - 0s 153us/sample - loss: 16.7053 - mean_absolute_error: 2.9730 - val_loss:
21.8842 - val_mean_absolute_error: 3.3088
Epoch 79/100
363/363 [==============================] - 0s 152us/sample - loss: 17.7637 - mean_absolute_error: 3.1019 - val_loss:
14.9161 - val_mean_absolute_error: 2.6454
Epoch 80/100
363/363 [==============================] - 0s 155us/sample - loss: 15.5705 - mean_absolute_error: 2.9974 - val_loss:
15.6584 - val_mean_absolute_error: 2.7240
Epoch 81/100
363/363 [==============================] - 0s 152us/sample - loss: 18.1282 - mean_absolute_error: 3.0500 - val_loss:
24.3890 - val_mean_absolute_error: 3.5217
Epoch 82/100
363/363 [==============================] - 0s 150us/sample - loss: 16.7609 - mean_absolute_error: 3.0855 - val_loss:
15.1565 - val_mean_absolute_error: 2.4849
Epoch 83/100
363/363 [==============================] - 0s 143us/sample - loss: 16.6251 - mean_absolute_error: 3.1269 - val_loss:
15.7590 - val_mean_absolute_error: 2.5150
Epoch 84/100
```

```
363/363 [==============================] - 0s 154us/sample - loss: 15.3808 - mean_absolute_error: 2.9848 - val_loss:
15.1085 - val_mean_absolute_error: 2.5136
Epoch 85/100
363/363 [==============================] - 0s 155us/sample - loss: 14.6282 - mean_absolute_error: 2.9044 - val_loss:
11.8692 - val_mean_absolute_error: 2.2236
Epoch 86/100
363/363 [==============================] - 0s 162us/sample - loss: 15.1583 - mean_absolute_error: 2.9104 - val_loss:
10.6420 - val_mean_absolute_error: 2.2027
Epoch 87/100
363/363 [==============================] - 0s 163us/sample - loss: 15.2363 - mean_absolute_error: 2.8730 - val_loss:
13.0868 - val_mean_absolute_error: 2.3120
Epoch 88/100
363/363 [==============================] - 0s 152us/sample - loss: 16.4723 - mean_absolute_error: 3.0707 - val_loss:
10.7341 - val_mean_absolute_error: 2.0796
Epoch 89/100
363/363 [==============================] - 0s 147us/sample - loss: 14.3021 - mean_absolute_error: 2.9648 - val_loss:
16.6098 - val_mean_absolute_error: 2.6837
Epoch 90/100
363/363 [==============================] - 0s 153us/sample - loss: 15.0252 - mean_absolute_error: 2.9164 - val_loss:
18.2483 - val_mean_absolute_error: 3.0696
Epoch 91/100
363/363 [==============================] - 0s 163us/sample - loss: 13.0592 - mean_absolute_error: 2.7966 - val_loss:
9.8557 - val_mean_absolute_error: 2.1037
Epoch 92/100
363/363 [==============================] - 0s 152us/sample - loss: 13.5856 - mean_absolute_error: 2.7889 - val_loss:
14.4998 - val_mean_absolute_error: 2.2953
Epoch 93/100
363/363 [==============================] - 0s 155us/sample - loss: 14.1522 - mean_absolute_error: 2.8078 - val_loss:
13.1824 - val_mean_absolute_error: 2.4106
Epoch 94/100
363/363 [==============================] - 0s 146us/sample - loss: 13.7745 - mean_absolute_error: 2.7760 - val_loss:
16.8860 - val_mean_absolute_error: 2.6348
Epoch 95/100
363/363 [==============================] - 0s 149us/sample - loss: 13.9821 - mean_absolute_error: 2.8292 - val_loss:
18.4003 - val_mean_absolute_error: 2.7871
Epoch 96/100
363/363 [==============================] - 0s 159us/sample - loss: 15.7507 - mean_absolute_error: 2.9404 - val_loss:
13.5345 - val_mean_absolute_error: 2.4566
Epoch 97/100
363/363 [==============================] - 0s 143us/sample - loss: 13.7275 - mean_absolute_error: 2.7779 - val_loss:
11.7003 - val_mean_absolute_error: 2.2706
Epoch 98/100
363/363 [==============================] - 0s 153us/sample - loss: 11.6555 - mean_absolute_error: 2.6346 - val_loss:
13.7839 - val_mean_absolute_error: 2.2545
Epoch 99/100
363/363 [==============================] - 0s 154us/sample - loss: 15.8250 - mean_absolute_error: 2.9396 - val_loss:
11.9409 - val_mean_absolute_error: 2.0516
Epoch 100/100
363/363 [==============================] - 0s 167us/sample - loss: 13.8763 - mean_absolute_error: 2.8265 - val_loss:
15.7340 - val_mean_absolute_error: 2.5169
```




```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class
'pandas.core.frame.DataFrame'>, <class 'NoneType'>
102/102 [==============================] - 0s 113us/sample - loss: 10.4099 - mean_absolute_error: 2.3375
損失: 10.409880198684393 MAE: 2.3375187
```

まとめ

ここまで、以下のことを学習しました。

- 教師あり学習の機械学習の流れ
    1. プロジェクトの目標を決める
    2. データを収集する
    3. データを分析する
    4. データを前処理する
    5. モデルを定義する
    6. 学習する
    7. 評価する
    8. チューニング
    9. モデルを保存する
    10. 推論
    11. 再学習
- データの前処理
    - 欠損値対応
      欠損値がある場合は、「0」などの値に変換したり、欠損値のあるデータそのものを除外します。
    - 多重共線性を防ぐ
      高い相関をもつ特徴量を使用して回帰を行うと、「多重共線性（multicollinearity、マルチコ）」が発生して学習できなくなることがあります。どちらかの特徴量を除外しましょう。
    - 訓練データ、検証データ、テストデータに分割する
    - 標準化
      値のとる範囲が広いと学習の収束が遅くなります。平均を0、分散1に変形することを「標準化」といいます。
- ニューラルネットワークモデルの作り方、使い方
    - モデルの定義
      モデルは層（レイヤー）を重ねて定義します。
    - 学習
      訓練データを `model.fit()` メソッドを使って学習させます。
    - 評価
      テストデータで `model.evaluate()` メソッドを使ってスコアを算出します。
    - 推論
      `model.predict()` メソッドを使って予測を行います。
- 回帰モデルの評価方法
  スカラー回帰（数値を予測する問題）では、MAE（平均絶対誤差）などを使って評価します。
- 作成したモデルの保存方法、保存したモデルの使い方
    - モデルの保存
      `model.save()` メソッドを使ってモデルを保存します。モデルの構造や、調整済みの重みが含まれます。
    - モデルの読み込み
      `models.load_model()` メソッドを使って、保存済みのモデルを読み込んで使います。
- 精度向上を目指して
    - データ数（パターン数）を増やす
    - モデルの改良
        - 隠れ層の出力ユニット数を変える
        - 隠れ層の数を変える
        - 最適化関数を変える
        - ドロップアウト層を追加する（過学習対策）
        - 正則化を行う（過学習対策）
    - ハイパーパラメータの改良
        - バッチサイズを変える
        - エポック数を変える
        - 最適化関数の学習率を変える

このお客さんは、商品を買う？買わない？（分類）

ある銀行では商品である定期預金のキャンペーンを行っています。顧客が契約してくれるかどうか分類してみましょう。
データセットはkaggleで公開されています。
https://www.kaggle.com/henriqueyamahata/bank-marketing

ここで学習すること

- 回帰と分類の違い
- one-hotベクトル
- 分類モデルの評価方法
- 不均衡データの解消方法

分類問題とは

分類問題とは、データがどのクラス（カテゴリ）に属するかを予測する問題です。
今回は、ある銀行で行われた定期預金商品のキャンペーンについてのデータセットを使って、顧客が「契約する（正：True）」「契約しない（負：False）」のどちらに属するかを分類してみましょう。
２つのクラスに分類する問題を二値分類、３つ以上のクラスに分類する問題を多クラス分類といいます。

回帰と分類の違い

回帰問題では、モデルは金額や数量など、そのものの値を出力していました。（22.5ドル、151個など）
分類問題では、人間があらかじめネコなら1、犬なら2を出力すると決めておき、モデルから2が出力されれば犬といった具合に分類を行います。（ニューラルネットワークは少し違い、それぞれのクラスに属する確率を出力します。）
言い換えれば、連続値を予想するのが回帰、離散値を予想するのが分類です。

目標の設定

二値分類の精度は以下のような指標で評価します。

- Accuracy（正解率） … 予測全体のうち、正しく予測できた割合。
- Precision（適合率）… 真と予測したデータのうち、実際に真である割合。
- Recall（再現率）… 実際に真であるデータのうち、真と予測した割合。
- Specificity（特異率）…実際に偽であるデータのうち、偽と予測した割合。Recall（再現率）の逆。
- F-score（F値） … Precision（適合率） と Recall（再現率） の調和平均。

通常はAccuracy（正解率）95%以上というような目標を設定しますが、目的によっては別の指標を重視します。

例）

- 健康診断で病気の人（真）と健康な人（偽）を分類する場合、本当に病気のある人を見逃したくないので、Recall（再現率）を重視します。
- メーラーでスパムメール（真）と正常のメール（偽）を分類する場合、正常なメールが迷惑フォルダーに入ると不便なため、Specificity（特異率）を重視します。

今回は商品を契約する顧客（真）と契約しない顧客（偽）に分類しますが、見込み客を見逃してセールスチャンスを失うことを避けたいため、「Accuracy（正解率）90%以上かつRecall（再現率）95%以上」のように目標を設定します。

ライブラリのインポート

プログラムで使用するライブラリをインポートします。
今回は以下のライブラリを使用します。

- numpy … NumPy(https://numpy.org/) 数値計算に使用します。
- pandas … pandas(https://pandas.pydata.org/) データの操作・統計に使用します。
- matplotlib … matplotlib(https://matplotlib.org/) グラフの描画に使用します。
- seaborn … seaborn(https://seaborn.pydata.org/) matplotlibのラッパーで、便利なグラフ描画機能が揃っています。
- tensorflow … TensorFlow(https://www.tensorflow.org/?hl=ja) ニューラルネットワークを使用するためのフレームワークです。
- tensorflow.keras … Keras(https://keras.io/ja/) TensorFlowのラッパーです。
- sklearn … scikit-learn(https://scikit-learn.org/stable/) 機械学習のライブラリです。

```
# matplotlibの最新バージョンを使用する
!pip install matplotlib --upgrade
```

```
Requirement already up-to-date: matplotlib in /usr/local/lib/python3.6/dist-packages (3.1.2)
Requirement already satisfied, skipping upgrade: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (0.10.0)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (2.6.1)
Requirement already satisfied, skipping upgrade: numpy>=1.11 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (1.17.4)
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matpl
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (1.1.0)
Requirement already satisfied, skipping upgrade: six in /usr/local/lib/python3.6/dist-packages (from cycler>=0.10->matplotlib) (1.12.0)
Requirement already satisfied, skipping upgrade: setuptools in /usr/local/lib/python3.6/dist-packages (from kiwisolver>=1.0.1->matplotlib) (41
```

```
# ライブラリのインポート
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

# TensorFlow 2.x系を使う
%tensorflow_version 2.x
import tensorflow as tf
from tensorflow.keras import models, layers

# TensorFlowのバージョン
print(tf.__version__)

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

# matplotlib の出力をノートブック上に描画するための設定
%matplotlib inline
```

```
TensorFlow 2.x selected.
2.0.0
```

データの読み込み

データはGoogleドライブに格納してありますので、Googleドライブをマウントして読み込みます。

```
# Google ドライブをマウント

from google.colab import drive

drive.mount('/content/drive')

# 表示されるURLにアクセスしてコードを取得し、下の入力欄に入力
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# データの格納パス
DATA_DIR = '/content/drive/My Drive/AI_lessons/2-2_2_classification_bank/data/'
# データのcsvファイル名
DATA_CSV = 'bank-additional-full.csv'
# 説明文のファイル名
NAME_TXT = 'bank-additional-names.txt'
```

このデータセットには説明文がついていますので、見てみましょう。

```
# 説明文
with open(DATA_DIR + NAME_TXT) as f:
  print(f.read())
```

```
Citation Request:
  This dataset is publicly available for research. The details are described in [Moro et al., 2014].
  Please include this citation if you plan to use this database:

  [Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Sys

  Available at: [pdf] http://dx.doi.org/10.1016/j.dss.2014.03.001
                [bib] http://www3.dsi.uminho.pt/pcortez/bib/2014-dss.txt

1. Title: Bank Marketing (with social/economic context)

2. Sources
   Created by: Sérgio Moro (ISCTE-IUL), Paulo Cortez (Univ. Minho) and Paulo Rita (ISCTE-IUL) @ 2014

3. Past Usage:

  The full dataset (bank-additional-full.csv) was described and analyzed in:

  S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems (2014), doi:10

4. Relevant Information:

   This dataset is based on "Bank Marketing" UCI dataset (please check the description at: http://archive.ics.uci.edu/ml/datasets/Bank+Marketi
   The data is enriched by the addition of five new social and economic features/attributes (national wide indicators from a ~10M population c
   This dataset is almost identical to the one used in [Moro et al., 2014] (it does not include all attributes due to privacy concerns).
   Using the rminer package and R tool (http://cran.r-project.org/web/packages/rminer/), we found that the addition of the five new social and

   The zip file includes two datasets:
      1) bank-additional-full.csv with all examples, ordered by date (from May 2008 to November 2010).
      2) bank-additional.csv with 10% of the examples (4119), randomly selected from bank-additional-full.csv.
   The smallest dataset is provided to test more computationally demanding machine learning algorithms (e.g., SVM).

   The binary classification goal is to predict if the client will subscribe a bank term deposit (variable y).

5. Number of Instances: 41188 for bank-additional-full.csv

6. Number of Attributes: 20 + output attribute.

7. Attribute information:

   For more information, read [Moro et al., 2014].

   Input variables:
   # bank client data:
   1 - age (numeric)
   2 - job : type of job (categorical: "admin.","blue-collar","entrepreneur","housemaid","management","retired","self-employed","services","stu
   3 - marital : marital status (categorical: "divorced","married","single","unknown"; note: "divorced" means divorced or widowed)
   4 - education (categorical: "basic.4y","basic.6y","basic.9y","high.school","illiterate","professional.course","university.degree","unknown"
   5 - default: has credit in default? (categorical: "no","yes","unknown")
   6 - housing: has housing loan? (categorical: "no","yes","unknown")
   7 - loan: has personal loan? (categorical: "no","yes","unknown")
   # related with the last contact of the current campaign:
   8 - contact: contact communication type (categorical: "cellular","telephone")
   9 - month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")
  10 - day_of_week: last contact day of the week (categorical: "mon","tue","wed","thu","fri")
  11 - duration: last contact duration, in seconds (numeric). Important note:  this attribute highly affects the output target (e.g., if durati
   # other attributes:
  12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
  13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not pre
  14 - previous: number of contacts performed before this campaign and for this client (numeric)
  15 - poutcome: outcome of the previous marketing campaign (categorical: "failure","nonexistent","success")
   # social and economic context attributes
  16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)
  17 - cons.price.idx: consumer price index - monthly indicator (numeric)
  18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)
  19 - euribor3m: euribor 3 month rate - daily indicator (numeric)
  20 - nr.employed: number of employees - quarterly indicator (numeric)

  Output variable (desired target):
  21 - y - has the client subscribed a term deposit? (binary: "yes","no")

8. Missing Attribute Values: There are several missing values in some categorical attributes, all coded with the "unknown" label. These missing
```

データセットについての説明と、各項目の説明が書かれていました。
各項目の内容は以下のとおりです。

No	項目	内容
# 顧客情報		

No	項目	内容
1	age	年齢。
2	job	職業。 "admin","blue-collar","entrepreneur","housemaid", "management","retired","self-employed","services", "student","technician","unemployed","unknown"
3	marital	結婚しているか。 "divorced","married","single","unknown"
4	education	学歴。 "basic.4y","basic.6y","basic.9y","high.school", "illiterate","professional.course","university.degree","unknown"
5	default	債務不履行か。 "no","yes","unknown"
6	housing	住宅ローンを契約しているか。 "no","yes","unknown"
7	loan	個人ローンを契約しているか。 "no","yes","unknown"
# このキャンペーンについて顧客に連絡した状況		
8	contact	連絡手段。 "cellular","telephone"
9	month	最後に連絡した月。 "jan", "feb", "mar", ..., "nov", "dec"
10	day_of_week	最後に連絡した曜日。 "mon","tue","wed","thu","fri"
11	duration	最後に連絡をした時の通話時間。 ※この項目は予測には使用しないこと。
12	campaign	連絡した回数。
13	pdays	前回のキャンペーンから何日後に連絡したか。 ※999は以前に一度も連絡をとっていない
14	previous	このキャンペーンの前に何回連絡をしたか。
15	poutcome	前回のキャンペーンの成約結果。 "failure","nonexistent","success"
# その他の指標		
16	emp.var.rate	四半期の雇用変動率
17	cons.price.idx	月の消費者物価指数
18	cons.conf.idx	月の消費者信頼感指数
19	euribor3m	3ヶ月間の欧州銀行間取引金利
20	nr.employed	四半期の社員数
# 出力		
21	y	顧客は定期預金を契約したか。"yes","no"

データファイルの内容を見てみましょう。

```
# データの先頭5行を表示
with open(DATA_DIR + DATA_CSV) as f:
  for i in range(5):
    print(f.readline())
```

```
"age";"job";"marital";"education";"default";"housing";"loan";"contact";"month";"day_of_week";"duration";"campaign";"pdays";"previous";"poutcome
56;"housemaid";"married";"basic.4y";"no";"no";"no";"telephone";"may";"mon";261;1;999;0;"nonexistent";1.1;93.994;-36.4;4.857;5191;"no"
57;"services";"married";"high.school";"unknown";"no";"no";"telephone";"may";"mon";149;1;999;0;"nonexistent";1.1;93.994;-36.4;4.857;5191;"no"
37;"services";"married";"high.school";"no";"yes";"no";"telephone";"may";"mon";226;1;999;0;"nonexistent";1.1;93.994;-36.4;4.857;5191;"no"
40;"admin.";"married";"basic.6y";"no";"no";"no";"telephone";"may";"mon";151;1;999;0;"nonexistent";1.1;93.994;-36.4;4.857;5191;"no"
```

1行目にはヘッダーがついており、各項目は「;」で区切られていることがわかりました。

`pd.read_csv()` を使うと、csvファイルからデータを読み込んで、DataFrame形式で扱うことができます。
カンマ以外の文字で区切られている場合は、 `sep` オプションで指定します。

```
#  csvファイルをDataFrameに読み込み
<p class="mume-header " id="csvファイルをdataframeに読み込み"></p>

df = pd.read_csv(DATA_DIR + DATA_CSV, sep=';')
df.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	camp
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	149	1
2	37	services	married	high.school	no	yes	no	telephone	may	mon	226	1
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1
4	56	services	married	high.school	no	no	yes	telephone	may	mon	307	1

データの分析

duration 列は契約の成否が判明した後に、分析用に作られた項目です。 y が no の場合は必ず 0 が格納されており（つまり未来の情報）、予測には使用できません。
duration 列は削除します。

```
#  duration 列を削除
<p class="mume-header " id="duration-列を削除"></p>

df.drop('duration', axis=1, inplace=True)
```

全般

データセット全般の情報を確認しましょう。

```
#  行数、列数
<p class="mume-header " id="行数-列数"></p>

df.shape
```

```
(41188, 20)
```

```
#  データ型
<p class="mume-header " id="データ型"></p>

df.dtypes
```

```
age               int64
job              object
marital          object
education        object
default          object
housing          object
loan             object
contact          object
month            object
day_of_week      object
campaign          int64
pdays             int64
previous          int64
poutcome         object
emp.var.rate    float64
cons.price.idx  float64
cons.conf.idx   float64
euribor3m       float64
nr.employed     float64
y                 object
dtype: object
```

```
#  pandasの表示の形式を指定
<p class="mume-header " id="pandasの表示の形式を指定"></p>

pd.options.display.float_format = '{:.2f}'.format

#  基本統計
<p class="mume-header " id="基本統計"></p>

df.describe()
```

	age	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
count	41188.00	41188.00	41188.00	41188.00	41188.00	41188.00	41188.00	41188.00	41188.00
mean	40.02	2.57	962.48	0.17	0.08	93.58	-40.50	3.62	5167.04
std	10.42	2.77	186.91	0.49	1.57	0.58	4.63	1.73	72.25
min	17.00	1.00	0.00	0.00	-3.40	92.20	-50.80	0.63	4963.60
25%	32.00	1.00	999.00	0.00	-1.80	93.08	-42.70	1.34	5099.10
50%	38.00	2.00	999.00	0.00	1.10	93.75	-41.80	4.86	5191.00
75%	47.00	3.00	999.00	0.00	1.40	93.99	-36.40	4.96	5228.10
max	98.00	56.00	999.00	7.00	1.40	94.77	-26.90	5.04	5228.10

`df.describe()` は、オプション `include` を指定しないと、数値型の列のみ表示します。
文字を含む列を表示するには、オプション `include='object'` をつけます。

```
# 基本統計（文字列型のカラム）
df.describe(include='object')
```

	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome	
count	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188
unique	12	4	8	3	3	3	2	10	5	3	2
top	admin.	married	university.degree	no	yes	no	cellular	may	thu	nonexistent	no
freq	10422	24928	12168	32588	21576	33950	26144	13769	8623	35563	3654

目的変数である y の値は、no の値が 36,548 件あることがわかりました。
つまり、 yes は残りの 4,640 件、全体の 1 割程度しか含まれていません。
データの分布に偏りがあるため、考慮が必要になりそうです。

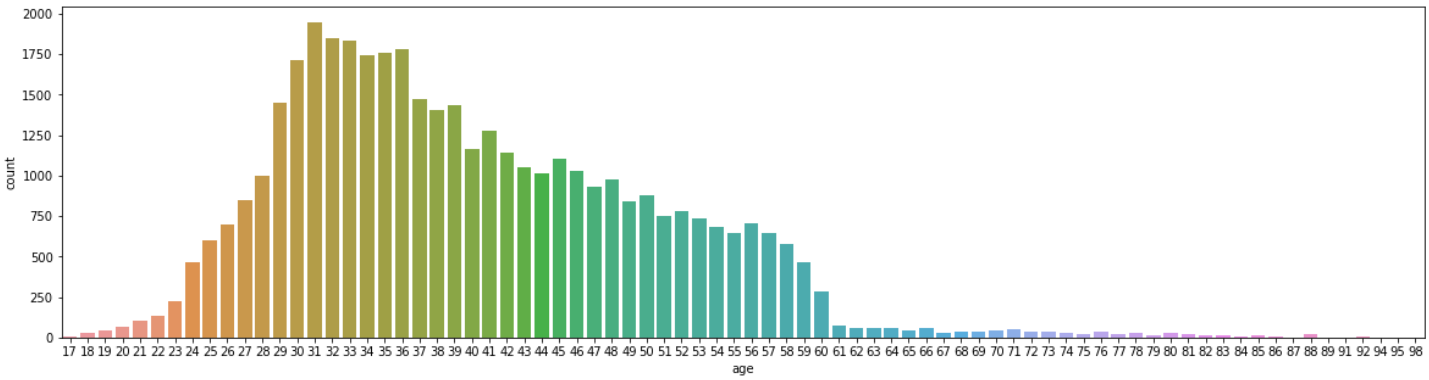
age

年齢について見てみましょう。
基本統計から、17〜98歳のデータが含まれることがわかっています。

データの分布を確認しましょう。
`sns.countplot()` でデータ件数をグラフ表示できます。

```
plt.figure(figsize=(20, 5))
sns.countplot(data=df, x='age')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c88045518>
```



30〜50代のデータが多いことがわかりました。
年齢は、目的変数 y とどの程度相関があるのでしょうか。

年齢では単位が細かいので、年代（10代、20代…）に変換します。

```
#  年代
<p class="mume-header " id="年代"></p>

df_age_category = df['age'].apply(lambda x: x // 10 * 10)
df_age_category.head()
```

```
0    50
1    50
2    30
3    40
4    50
Name: age, dtype: int64
```

年代と y をクロス集計します。
`pd.crosstab()` を使うと簡単にクロス集計を行うことができます。
`normalize` オプションを指定して正規化を行うことができます。'index'を指定すると、行の合計が1になるように正規化されます。

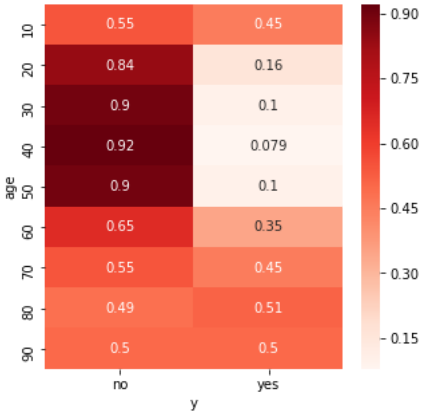
```
#  年代と y をクロス集計
<p class="mume-header " id="年代と-y-をクロス集計"></p>

df_cross = pd.crosstab(df_age_category, df['y'], normalize='index')
df_cross
```

y	no	yes
age		
10	0.55	0.45
20	0.84	0.16
30	0.90	0.10
40	0.92	0.08
50	0.90	0.10
60	0.65	0.35
70	0.55	0.45
80	0.49	0.51
90	0.50	0.50

数字だけでは分かりづらいので、ヒートマップにしてみましょう。

```
#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画"></p>

plt.figure(figsize=(5, 5))
sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c87704d68>
```



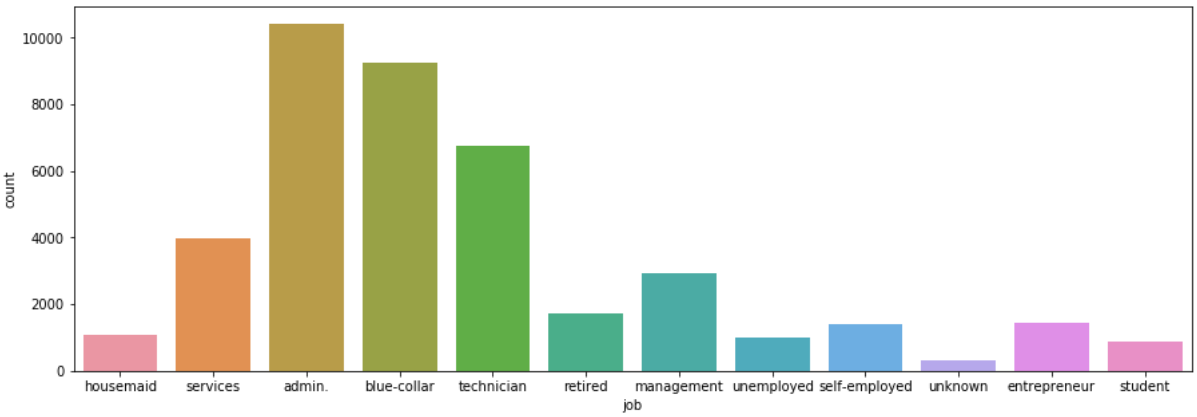
10代と、60～90代が多く契約しています。
一方、20代～50代は あまり契約していません。

job

職業について見てみましょう。

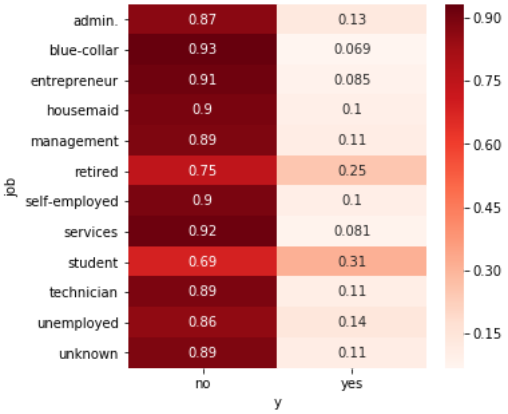
```
plt.figure(figsize=(15, 5))
sns.countplot(data=df, x='job')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c87b706d8>
```




```
#  クロス集計
<p class="mume-header " id="クロス集計"></p>

df_cross = pd.crosstab(df['job'], df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-1"></p>

plt.figure(figsize=(5, 5))
sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c83aa7278>
```

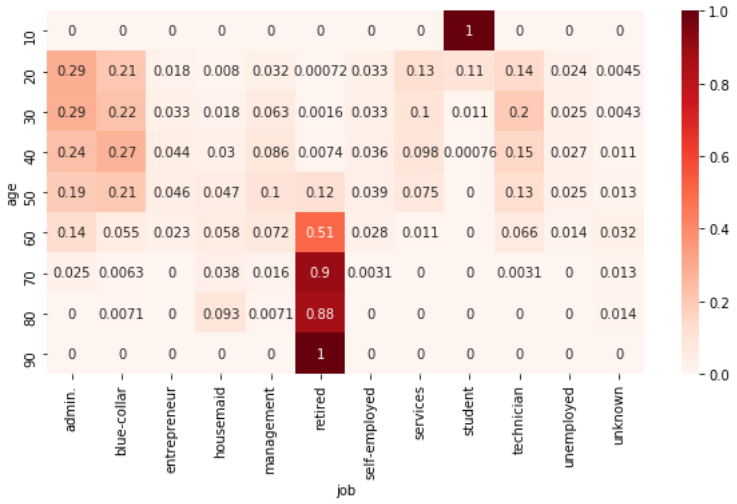


yes の割合が高いのは、退職者と学生です。

学生や退職者といった職業は、年齢と関係がありそうです。
職業を年代別にクロス集計してみます。

```
#  年代と職業をクロス集計
<p class="mume-header " id="年代と職業をクロス集計"></p>

df_cross = pd.crosstab(df_age_category, df['job'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-2"></p>

plt.figure(figsize=(10, 5))
sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c83a4c9e8>
```



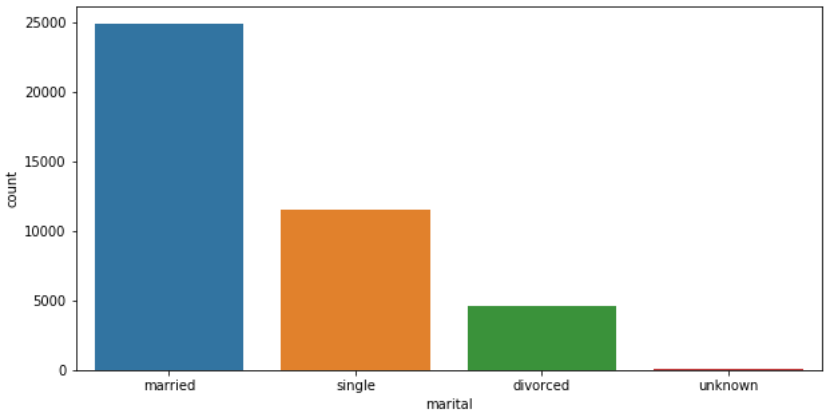
予想通り、退職者と学生は年齢と高い相関があることがわかりました。

marital

結婚しているかどうかは関係があるでしょうか。

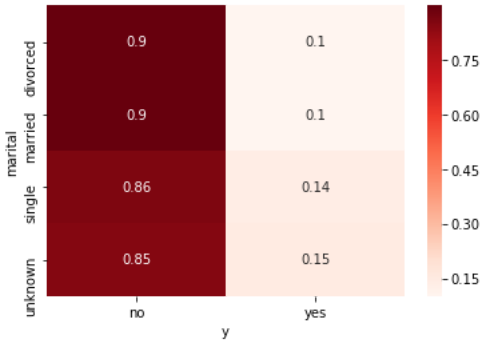
```
#  データの割合
<p class="mume-header " id="データの割合"></p>

plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='marital')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c877046d8>
```




```
#  クロス集計
<p class="mume-header " id="クロス集計-1"></p>

df_cross = pd.crosstab(df['marital'], df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-3"></p>

sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c83829048>
```



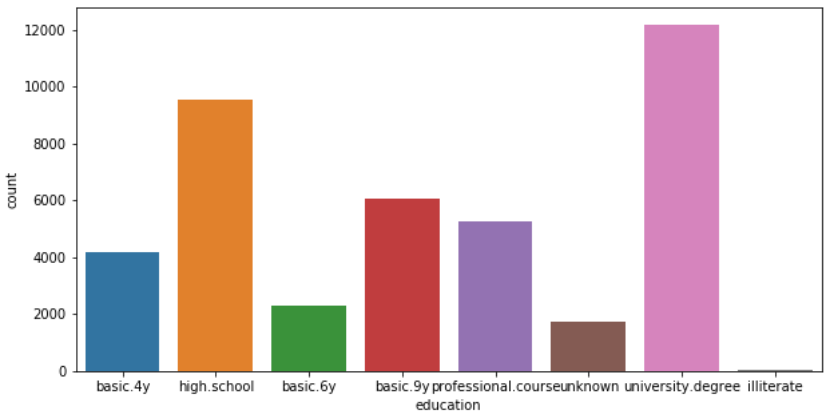
わずかに single と unknown が定期預金を契約する割合が高いようです。

education

学歴について見てみましょう。

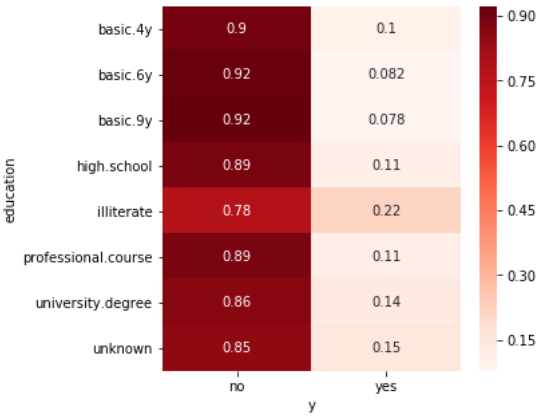
```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='education')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c837c2fd0>
```



```
#  クロス集計
<p class="mume-header " id="クロス集計-2"></p>

df_cross = pd.crosstab(df['education'], df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-4"></p>

plt.figure(figsize=(5, 5))
sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c837c2dd8>
```



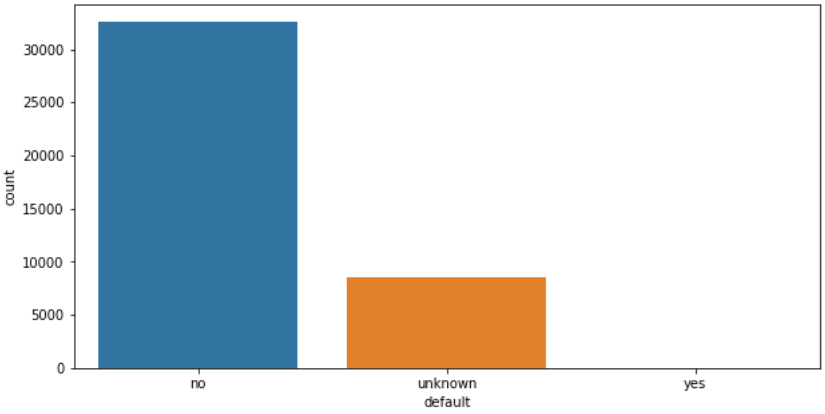
illiterate（教育を受けていない人）がわずかに yes の割合が高いです。

default

債務不履行かどうかを見てみましょう。

```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='default')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c83796048>
```



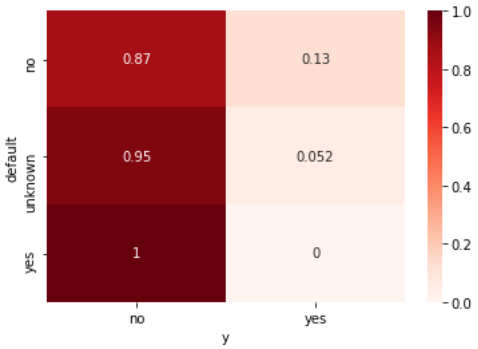
default が yes のデータはとても少ないです。
どんなデータか見てみましょう。

```
#  Default = yes のデータ
<p class="mume-header " id="default-yes-のデータ"></p>

df[df['default']=='yes']
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	campai
21580	48	technician	married	professional.course	yes	no	no	cellular	aug	tue	1
21581	48	technician	married	professional.course	yes	yes	no	cellular	aug	tue	1
24866	31	unemployed	married	high.school	yes	no	no	cellular	nov	tue	2

```
#  クロス集計
<p class="mume-header " id="クロス集計-3"></p>

df_cross = pd.crosstab(df['default'], df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-5"></p>

sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c83632b70>
```



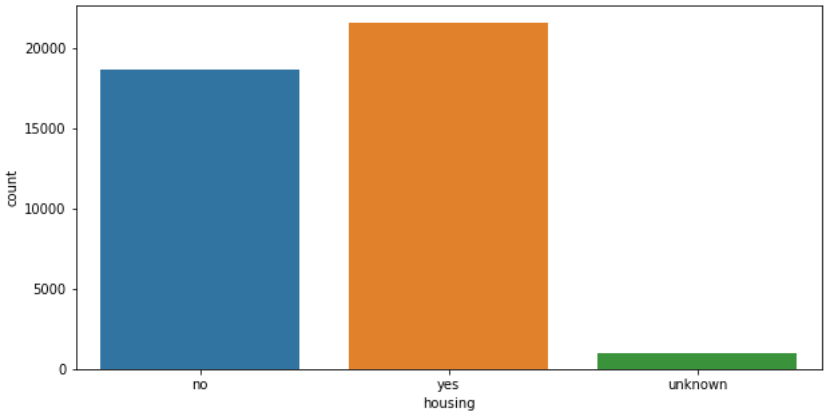
債務不履行になっていない人は unknown に比べて、定期預金を契約する割合が高いです。
一方、債務不履行の人は全く契約をしていませんが、母数が3件ですのでサンプルが十分とは言えません。

housing

住宅ローンの有無は関係があるでしょうか。

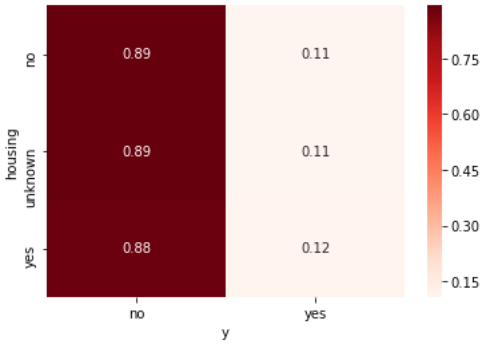
```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='housing')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3ccfdeacc0>
```




```
#  クロス集計
<p class="mume-header " id="クロス集計-4"></p>

df_cross = pd.crosstab(df['housing'], df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-6"></p>

sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c83655fd0>
```



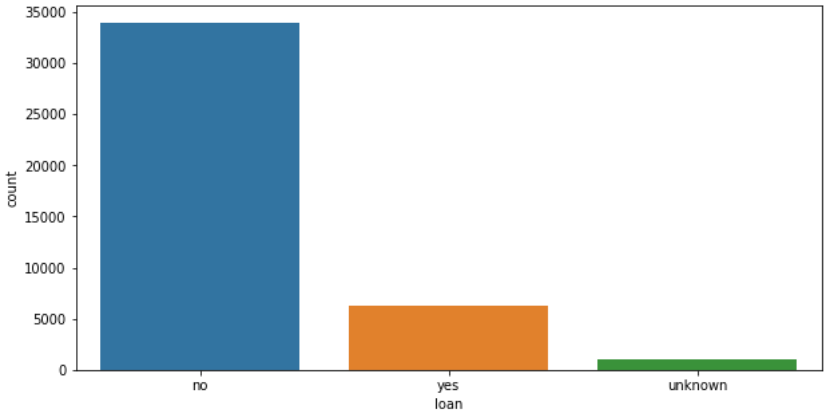
住宅ローンの有無は関係なさそうです。

loan

個人ローンの有無は関係があるでしょうか。

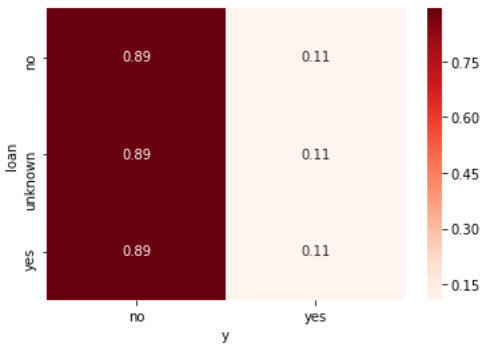
```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='loan')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c834bc438>
```




```
#  クロス集計
<p class="mume-header " id="クロス集計-5"></p>

df_cross = pd.crosstab(df['loan'], df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-7"></p>

sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c83438cc0>
```



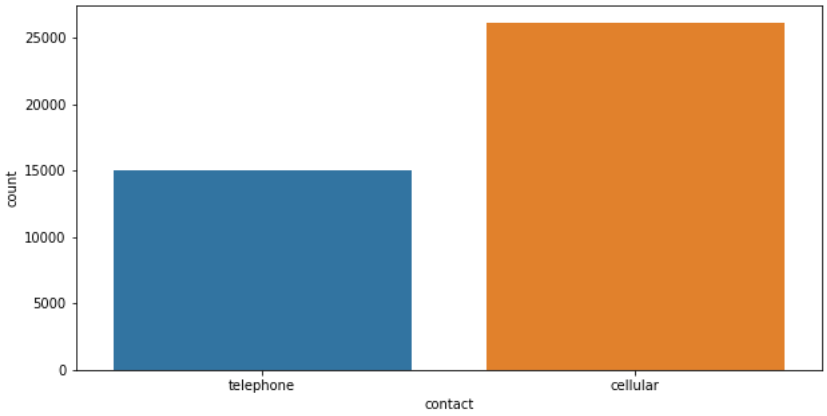
個人ローンの有無は関係なさそうです。

contact

連絡手段によって、契約の成否は変わるのでしょうか。

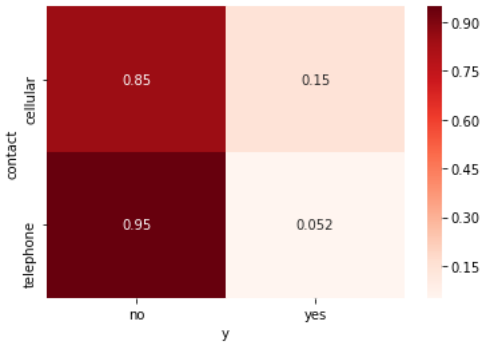
```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='contact')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c8356c550>
```




```
#  クロス集計
<p class="mume-header " id="クロス集計-6"></p>

df_cross = pd.crosstab(df['contact'], df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-8"></p>

sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c83389320>
```



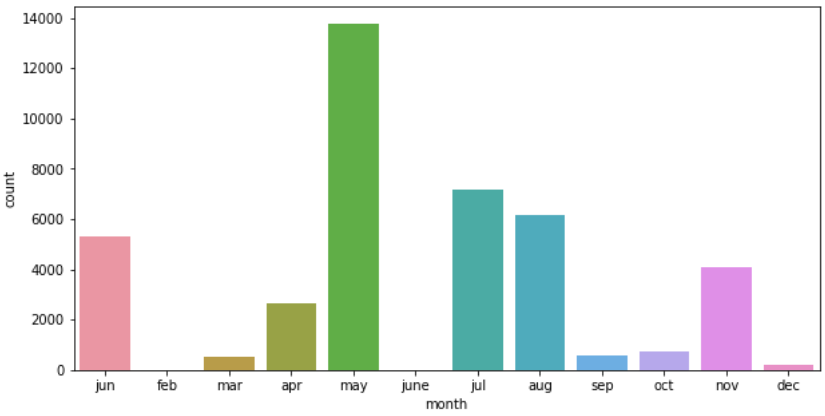
cellular の方が、 telephone の 3 倍契約に結びついていることがわかりました。

month

キャンペーンを行った月は関係があるでしょうか。

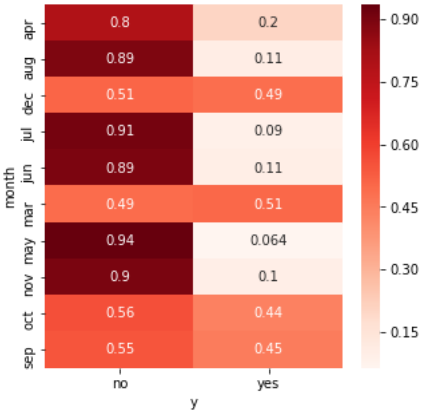
```
month_names = ['jun', 'feb', 'mar', 'apr', 'may', 'june', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec']
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='month', order=month_names)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c8356c518>
```




```
#  クロス集計
<p class="mume-header " id="クロス集計-7"></p>

df_cross = pd.crosstab(df['month'], df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-9"></p>

plt.figure(figsize=(5, 5))
sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c833d54a8>
```



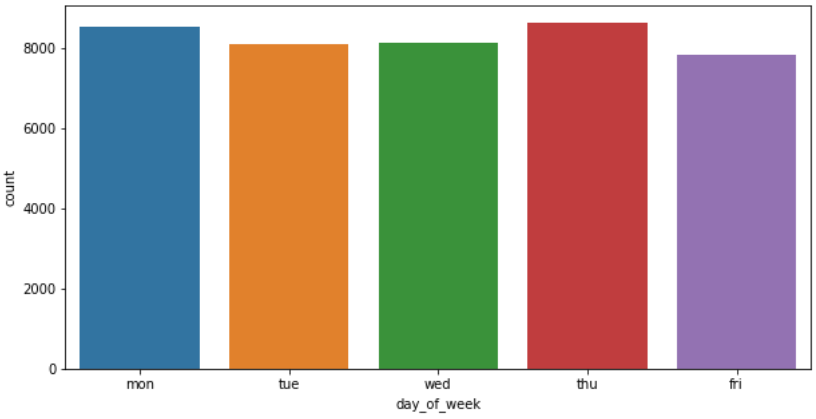
3月、7月、8月、12月は契約する割合が高いようです。

day of week

キャンペーンで連絡をとる曜日には関係があるでしょうか。

```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='day_of_week')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c831d5ef0>
```



```
#  クロス集計
<p class="mume-header " id="クロス集計-8"></p>

df_cross = pd.crosstab(df['day_of_week'], df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-10"></p>

sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c830f6f28>
```

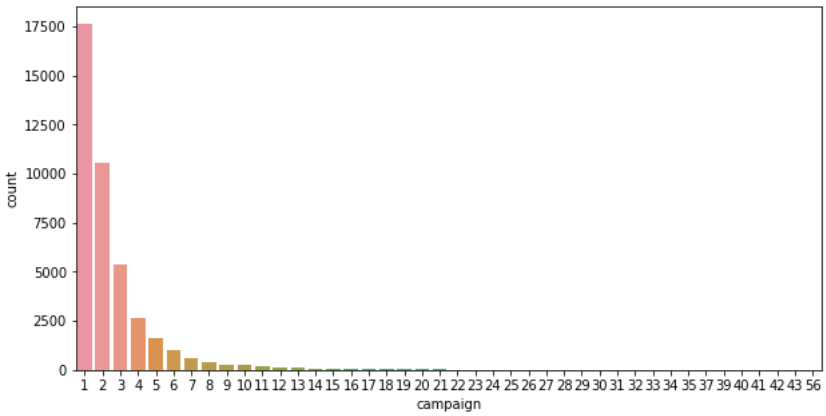
ごく僅かですが、月曜日は契約する割合が低いです。

campaign

このキャンペーンで連絡をとった回数は関係あるでしょうか。

```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='campaign')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c83084b00>
```



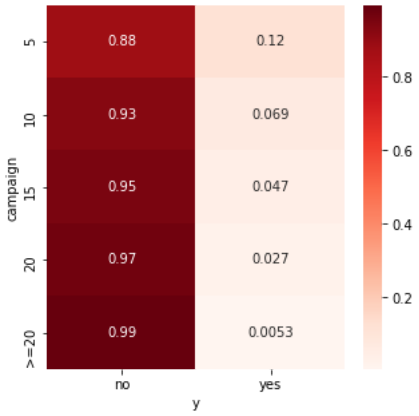
22回以上はデータ数が少ないです。

```
#  22回以上連絡をとった件数
<p class="mume-header " id="22回以上連絡をとった件数"></p>

df[df['campaign']>=22]['age'].count()
```

```
133
```

```
#  単位が細かいので 5 回刻みに変換（20回以上は「>=20」に集計）
<p class="mume-header " id="単位が細かいので-5-回刻みに変換20回以上は20に集計"></p>

df_campaign_category = df['campaign'].apply(lambda x: (x // 5 + 1) * 5 if x < 20 else '>=20')

#  クロス集計
<p class="mume-header " id="クロス集計-9"></p>

df_cross = pd.crosstab(df_campaign_category, df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-11"></p>

plt.figure(figsize=(5, 5))
sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c82f28978>
```



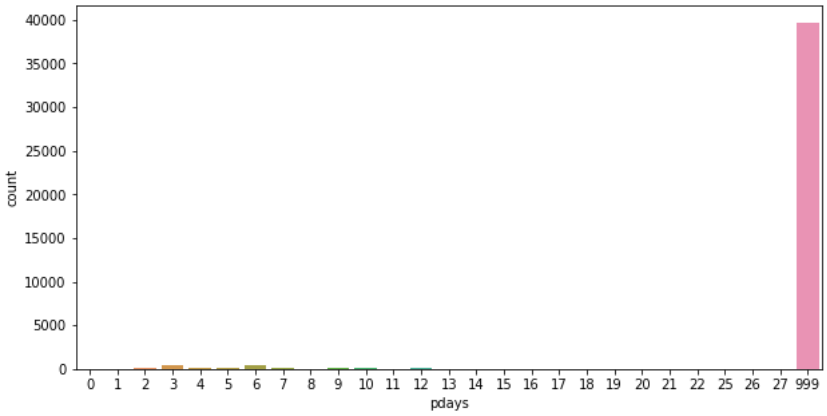
連絡回数が増えるほど、契約に結びつく可能性は低くなっています。

pdays

前回のキャンペーンから何日後に連絡したかは、関係があるでしょうか。

```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='pdays')
```

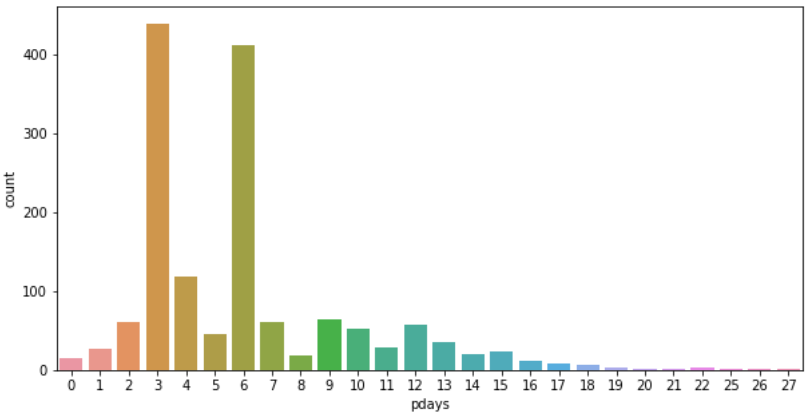
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c82eeaf28>
```



「999」は前回のキャンペーンでは連絡をとらなかった顧客です。
圧倒的に「999」の件数が多く、他のデータの分布が分かりづらいので、「999」を除いてみます。

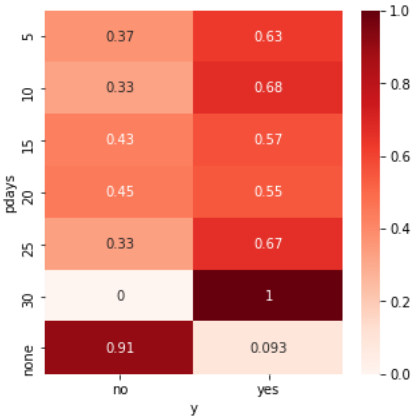
```
#  999を除くデータの分布
<p class="mume-header " id="999を除くデータの分布"></p>

plt.figure(figsize=(10, 5))
sns.countplot(data=df[df['pdays']!=999], x='pdays')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c82e19588>
```




```
#  単位が細かいので 5 回刻みに変換。「999」は'none'
<p class="mume-header " id="単位が細かいので-5-回刻みに変換999はnone"></p>

df_pdays_category = df['pdays'].apply(lambda x: (x // 5 + 1) * 5 if x != 999 else 'none')

#  クロス集計
<p class="mume-header " id="クロス集計-10"></p>

df_cross = pd.crosstab(df_pdays_category, df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-12"></p>

plt.figure(figsize=(5, 5))
sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c82cabd30>
```



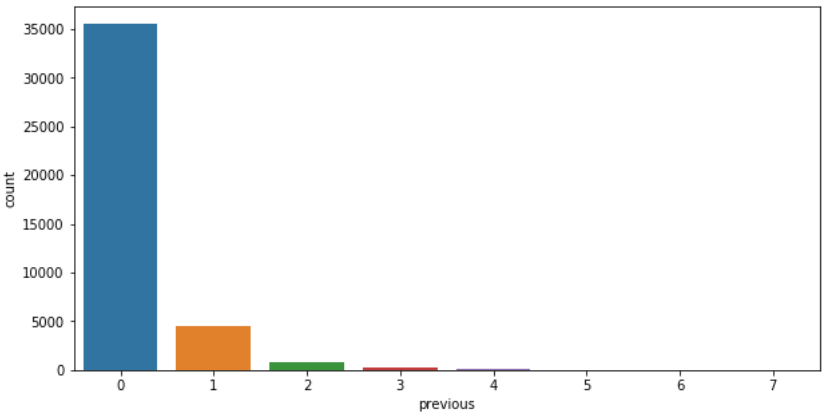
前回のキャンペーンで連絡をとったか、とっていないかで、はっきりと傾向が別れています。

previous

このキャンペーンの前に何回連絡をしたかを見てみましょう。

```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='previous')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c835bbd30>
```

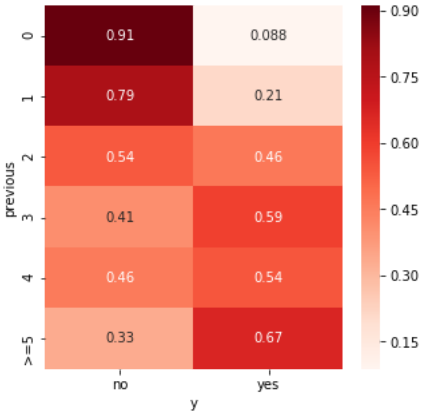



```
#  previousが5回以上の件数
<p class="mume-header " id="previousが5回以上の件数"></p>

df[df['previous']>=5]['age'].count()
```

```
24
```

```
#  5以上はひとまとめにする
<p class="mume-header " id="5以上はひとまとめにする"></p>

df_previous_category = df['previous'].apply(lambda x: x if x < 5 else '>=5')

#  クロス集計
<p class="mume-header " id="クロス集計-11"></p>

df_cross = pd.crosstab(df_previous_category, df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-13"></p>

plt.figure(figsize=(5, 5))
sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c82c01be0>
```



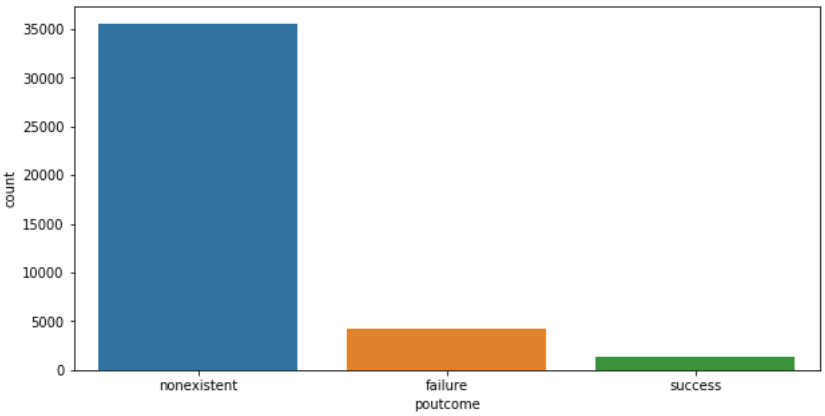
連絡の回数が多いほど、yes の割合が高いようです。

poutcome

前回のキャンペーンの契約成否は関係があるでしょうか。

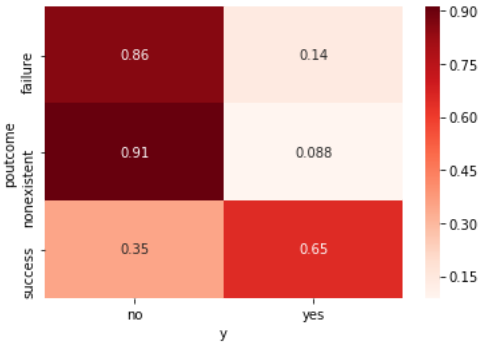
```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='poutcome')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c82cb9e10>
```




```
#  クロス集計
<p class="mume-header " id="クロス集計-12"></p>

df_cross = pd.crosstab(df['poutcome'], df['y'], normalize='index')

#  ヒートマップ描画
<p class="mume-header " id="ヒートマップ描画-14"></p>

sns.heatmap(df_cross, annot=True, cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c82ab1e80>
```



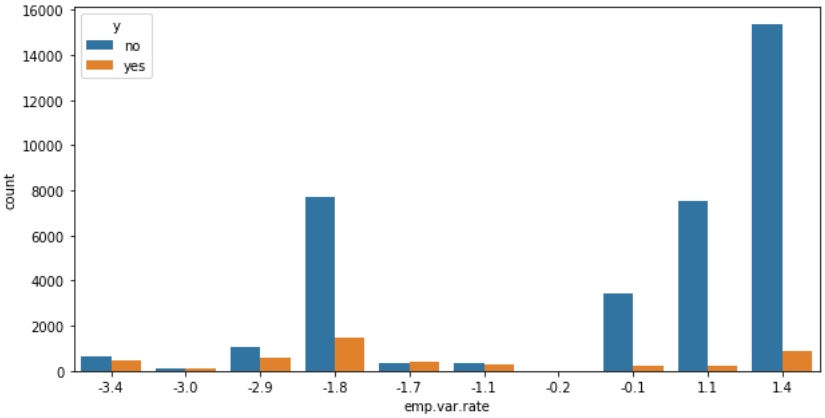
前回契約した顧客は、今回も契約する確率が高いことがわかりました。

emp.var.rate

ここからは外部要因の指標です。四半期の雇用変動率を見てみましょう。

```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='emp.var.rate', hue='y')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c830eb908>
```



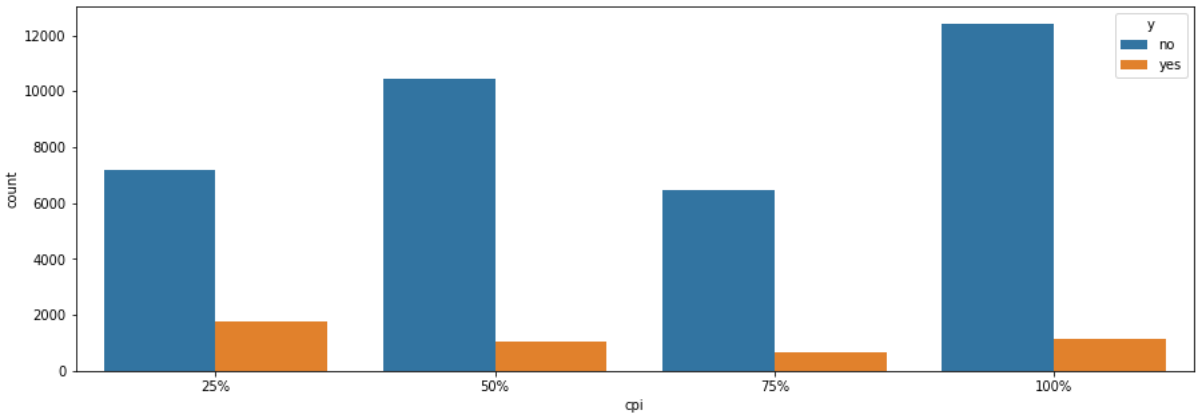
emp.var.rate がマイナスの時に、定期預金を契約する割合が高いようですが、一概にそうとも言い切れません。

cons.price.idx

消費者物価指数です。

cons.price.idx はかなり細かい値が格納されていますので、四分位数で分けてみます。

```
def to_category_quantile(x, q):
  if x < q[0.25]:
    return '25%'
  elif x < q[0.5]:
    return '50%'
  elif x < q[0.75]:
    return '75%'
  else:
    return '100%'


#  四分位数
<p class="mume-header " id="四分位数"></p>

q = df['cons.price.idx'].quantile([0.25, 0.5, 0.75])

df_cpi_category = pd.DataFrame()
df_cpi_category['cpi'] = df['cons.price.idx'].apply(lambda x: to_category_quantile(x, q))
df_cpi_category['y'] = df['y']


plt.figure(figsize=(15, 5))
sns.countplot(data=df_cpi_category, x='cpi', hue='y', order=['25%', '50%', '75%', '100%'])


<matplotlib.axes._subplots.AxesSubplot at 0x7f3c82923e48>
```

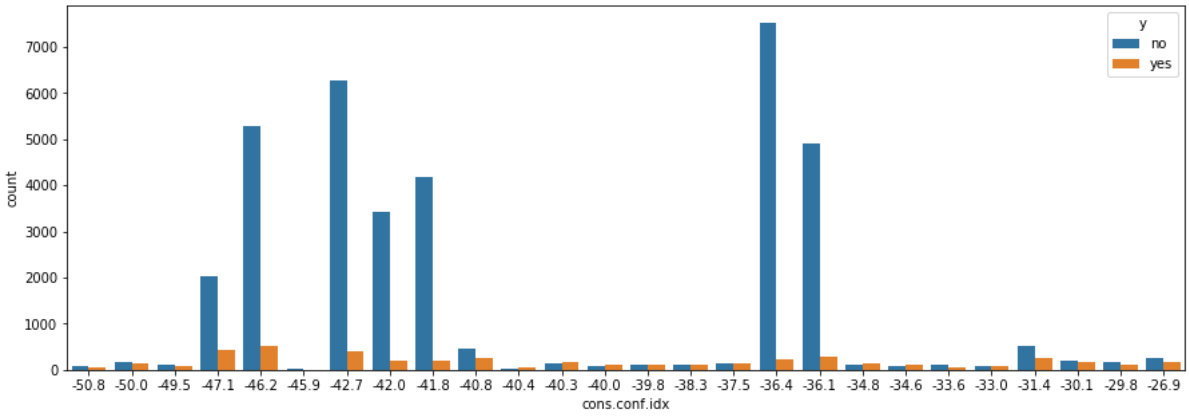


値が小さい方（物価が安い方）が、定期預金を契約する割合が高いです。

cons.conf.idx

消費者信頼感指標です。

```
plt.figure(figsize=(15, 5))
sns.countplot(data=df, x='cons.conf.idx', hue='y')


<matplotlib.axes._subplots.AxesSubplot at 0x7f3c82987ef0>
```

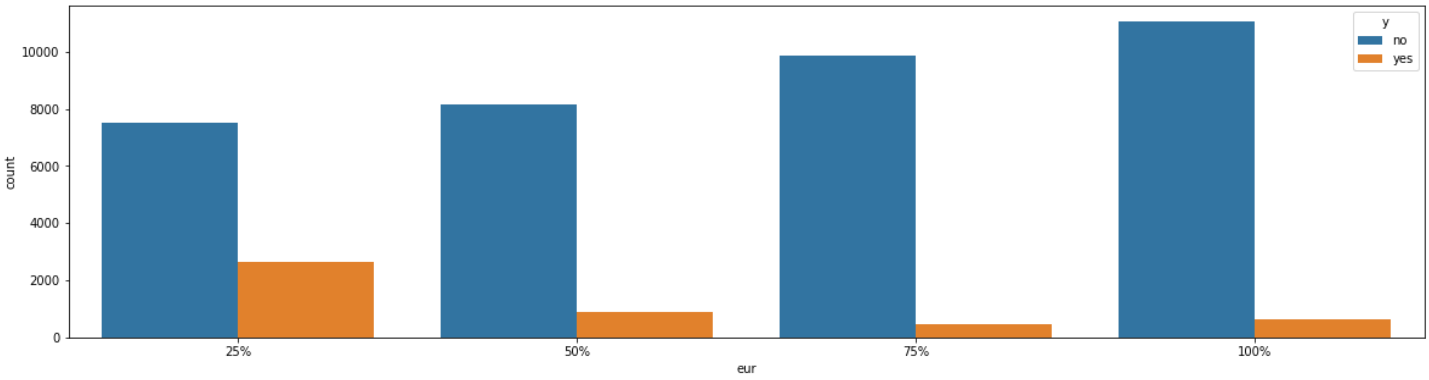


cons.conf.idxと y には、特に相関はみられません。

euribor3m

欧州銀行間取引金利です。

```
#  四分位数
<p class="mume-header " id="四分位数-1"></p>

q = df['euribor3m'].quantile([0.25, 0.5, 0.75])

df_eur_category = pd.DataFrame()
df_eur_category['eur'] = df['euribor3m'].apply(lambda x: to_category_quantile(x, q))
df_eur_category['y'] = df['y']


plt.figure(figsize=(20, 5))
sns.countplot(data=df_eur_category, x='eur', hue='y', order=['25%', '50%', '75%', '100%'])


<matplotlib.axes._subplots.AxesSubplot at 0x7f3c82960080>
```

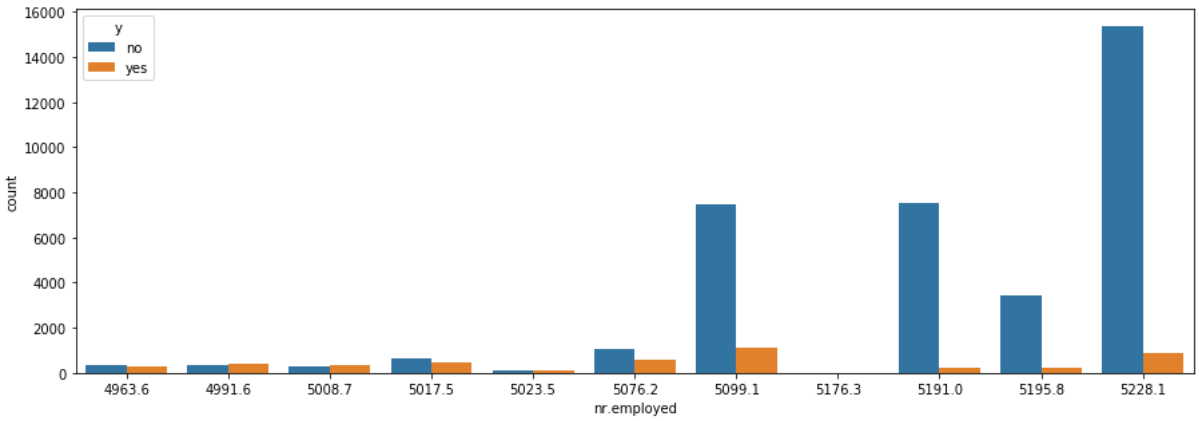


euribor が低いほど、契約する割合が高いです。

nr.employed

四半期の従業員数です。

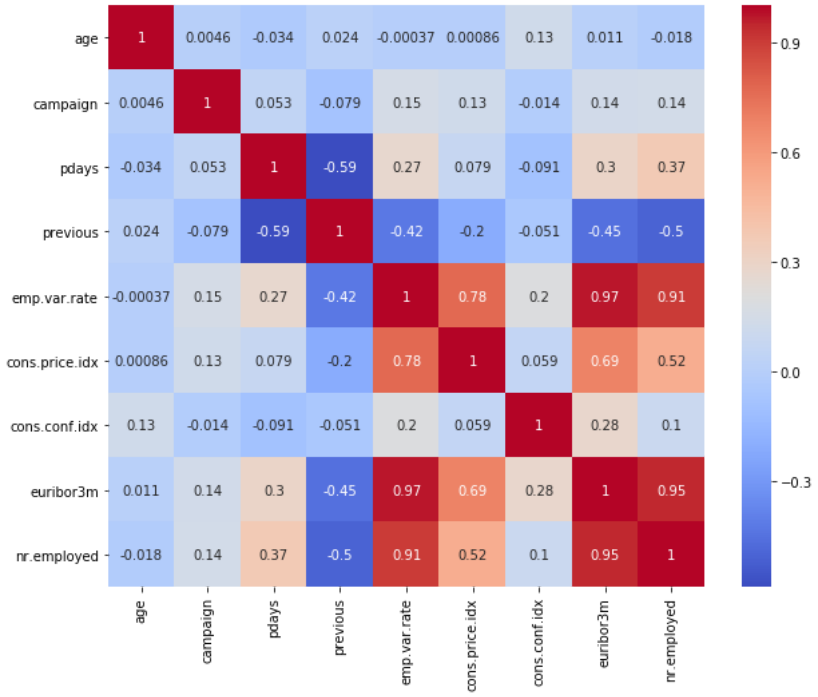
```
plt.figure(figsize=(15, 5))
sns.countplot(data=df, x='nr.employed', hue='y')


<matplotlib.axes._subplots.AxesSubplot at 0x7f3c832a8048>
```



従業員数が少ない時の方が契約する割合が高いという結果になっています。
ですが、従業員数と契約の成否が直接関係しているとは考えにくいです。
従業員数は他の特徴料の代替変数である可能性を考えて、特徴量間の相関を見てみましょう。

```
#  相関係数を計算
<p class="mume-header " id="相関係数を計算"></p>

df_corr = df.corr()

#  ヒートマップを描画
<p class="mume-header " id="ヒートマップを描画"></p>

plt.figure(figsize=(10, 8))
sns.heatmap(df_corr, annot=True, cmap='coolwarm')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c826ad0b8>
```



emp.var.rate、euribor3m、nr.employedは強い相関があることがわかりました。
emp.var.rate、nr.employedはeuribor3mの代替変数の可能性が高いです。

前処理

特徴量を選択する

y と無相関だった項目や、特徴量同士の相関が高い項目は除きます。

```
df_wk = pd.DataFrame(df[['age', 'job', 'marital', 'education', 'default',
                         'contact', 'month', 'day_of_week', 'campaign',
                         'pdays', 'previous', 'poutcome',
                         'cons.price.idx', 'euribor3m',
                         'y']])
df_wk.head()
```

	age	job	marital	education	default	contact	month	day_of_week	campaign	pdays	previous	p
0	56	housemaid	married	basic.4y	no	telephone	may	mon	1	999	0	no
1	57	services	married	high.school	unknown	telephone	may	mon	1	999	0	no
2	37	services	married	high.school	no	telephone	may	mon	1	999	0	no
3	40	admin.	married	basic.6y	no	telephone	may	mon	1	999	0	no
4	56	services	married	high.school	no	telephone	may	mon	1	999	0	no

データの整形

pdays は 999 が特殊な意味をもつ値でした。
データの分析結果から「999 か、それ以外か」を表す値に変換します。

```
#  pdays 999→0、それ以外→1
<p class="mume-header " id="pdays-9990-それ以外1"></p>

df_wk['pdays'] = df_wk['pdays'].apply(lambda x: 1 if x != 999 else 0)
```

ダミー変数化（One-hotベクトル変換）

カテゴリ変数はそのまま機械学習で扱うことはできませんので、ダミー変数（One-hotベクトル）に変換します。
One-hotベクトルとは、それぞれのカテゴリについて、該当するものにだけ1、他は0を格納したリストです。
pandas.DataFrameでは `pd.get_dummies()` で簡単にダミー変数に変換できます。

jobを例に見てみましょう。以下のようなデータがあったとします。

job
student
housemaid
services

`pd.get_dummies()` を使うと、以下のように変換されます。

job_student	job_housemaid	job_services
1	0	0
0	1	0
0	0	1

```
# ダミー変数化
df_wk = pd.get_dummies(df_wk, columns=['job', 'marital', 'education', 'default', 'contact', 'month', 'day_of_week', 'pdays', 'poutcome'])
df_wk.head()
```

	age	campaign	previous	cons.price.idx	euribor3m	y	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid
0	56	1	0	93.99	4.86	no	0	0	0	1
1	57	1	0	93.99	4.86	no	0	0	0	0
2	37	1	0	93.99	4.86	no	0	0	0	0
3	40	1	0	93.99	4.86	no	1	0	0	0
4	56	1	0	93.99	4.86	no	0	0	0	0

訓練データ・検証データ・テストデータに分割する

データを、訓練データ、検証データ、テストデータに分割します。

今回のデータは y = yes の件数が少ないため、yes のデータが偏らないよう、yes と noをそれぞれ分けて分割します。

```
# yesとnoに分離
df_yes = df_wk[df_wk['y'] == 'yes']
df_no = df_wk[df_wk['y'] == 'no']

# 訓練データ8割、評価データ2割に分割
train_yes, test_yes = train_test_split(df_yes, test_size=0.2, random_state=777)
train_no, test_no = train_test_split(df_no, test_size=0.2, random_state=777)

# 更に訓練データ8割、検証データ2割に分割
train_yes, val_yes = train_test_split(train_yes, test_size=0.2, random_state=777)
train_no, val_no = train_test_split(train_no, test_size=0.2, random_state=777)

# yes と no を結合、sample(frac=1)でシャッフル
df_train = pd.concat([train_yes, train_no]).sample(frac=1)
df_val = pd.concat([val_yes, val_no]).sample(frac=1)
df_test = pd.concat([test_yes, test_no]).sample(frac=1)
```

データとラベルに分ける

特徴量データと、ラベル（y）に分けます。

```
# データとラベルに分離
x_train = pd.DataFrame(df_train.drop('y', axis=1))
y_train = pd.DataFrame(df_train['y'])
x_val = pd.DataFrame(df_val.drop('y', axis=1))
y_val = pd.DataFrame(df_val['y'])
x_test = pd.DataFrame(df_test.drop('y', axis=1))
y_test = pd.DataFrame(df_test['y'])

print('x_train shape', x_train.shape)
print('y_train shape', y_train.shape)
print('x_val shape', x_val.shape)
print('y_val shape', y_val.shape)
print('x_test shape', x_test.shape)
print('y_test shape', y_test.shape)
```

```
x_train shape (26359, 54)
y_train shape (26359, 1)
x_val shape (6591, 54)
y_val shape (6591, 1)
x_test shape (8238, 54)
y_test shape (8238, 1)
```

ラベルはyes/noのカテゴリ変数ですので、ダミー変数（One-hotベクトル）に変換します。

```
# one-hotベクトルに変換
y_train = pd.get_dummies(y_train['y'])
y_val = pd.get_dummies(y_val['y'])
y_test = pd.get_dummies(y_test['y'])

y_train.head()
```

	no	yes
24158	1	0
30159	0	1
17620	1	0
26196	1	0
28987	1	0

後述しますが、推論の結果はリストで出力されます。
リストの順番は、カラムの順番と同じですので、

```
[0] … no
[1] … yes
```

に対応します。

標準化

標準化を行います。

```
# 各列の平均
mean = x_train.mean()
# 各列の標準偏差
std = x_train.std(ddof=False)

# 平均を引き、標準偏差で割る
x_train -= mean
x_train /= std

x_train.describe()
```

	age	campaign	previous	cons.price.idx	euribor3m	job_admin.	job_blue-collar	job_entrepreneur	job_housem
count	26359.00	26359.00	26359.00	26359.00	26359.00	26359.00	26359.00	26359.00	26359.00

	age	campaign	previous	cons.price.idx	euribor3m	job_admin.	job_blue-collar	job_entrepreneur	job_housem
mean	-0.00	0.00	-0.00	0.00	-0.00	0.00	-0.00	0.00	0.00
std	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
min	-2.21	-0.57	-0.35	-2.37	-1.72	-0.58	-0.54	-0.19	-0.16
25%	-0.77	-0.57	-0.35	-0.86	-1.31	-0.58	-0.54	-0.19	-0.16
50%	-0.19	-0.20	-0.35	0.38	0.71	-0.58	-0.54	-0.19	-0.16
75%	0.67	0.16	-0.35	0.72	0.77	1.72	-0.54	-0.19	-0.16
max	5.00	19.31	13.70	2.05	0.82	1.72	1.86	5.20	6.08

```
# 検証データ、テストデータも同様に処理する
x_val -= mean
x_val /= std

x_test -= mean
x_test /= std
```

モデルの定義

モデルを定義します。
回帰のモデルと異なるのは、最後の出力層です。
回帰ではユニット数は1、活性化関数は使用しませんでした。
分類では、出力層のユニット数は、分類したいクラスの数を指定します。今回は yes / no の2クラスですので、「2」を指定します。
また、活性化関数は、yes / no の確率を足して 1 になるよう、「softmax」を使用します。

```
# モデルの定義
model = tf.keras.Sequential([
  layers.Dense(16, activation='relu', input_shape=(len(x_train.columns), )),
  layers.Dropout(0.2),
  layers.Dense(16, activation='relu'),
  layers.Dropout(0.1),
  layers.Dense(2, activation='softmax')
])

# モデルの構造情報
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                880
_________________________________________________________________
dropout (Dropout)            (None, 16)                0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_1 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 34
=================================================================
Total params: 1,186
Trainable params: 1,186
Non-trainable params: 0
_________________________________________________________________
```

モデルをコンパイルします。
カテゴリ分類の場合、損失関数は「categorical_crossentropy」を使用します。
評価関数は「accuracy（正解率）」を指定します。

```
# モデルのコンパイル
model.compile(optimizer='adam',                    # 最適化アルゴリズム…Adam
              loss='categorical_crossentropy',     # 損失関数…Categorical Crossentropy
              metrics=['accuracy'])                # 評価関数…正解率
```

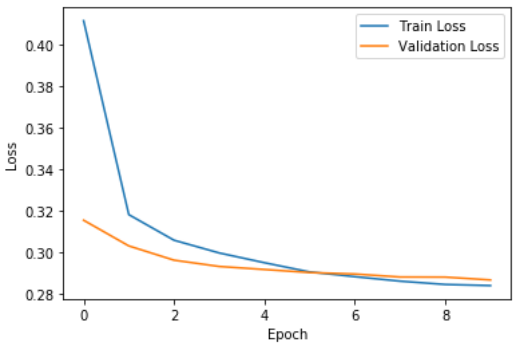
学習

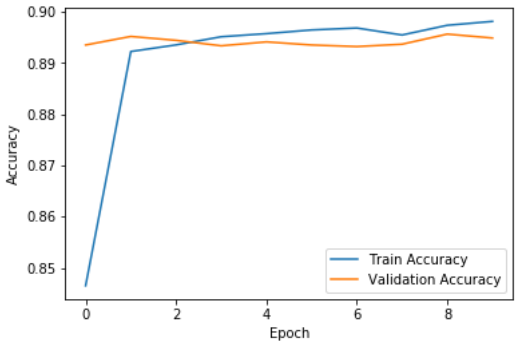
訓練データを使って学習します。

```
# 学習とエポックごとの評価
history = model.fit(x_train,                            # 訓練データ
                    y_train,                            # 訓練データのラベル
                    batch_size=128,                     # バッチサイズ
                    epochs=10,                          # エポック数
                    validation_data=(x_val, y_val))     # 検証データ
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class 'pandas.core.frame.Dat
Train on 26359 samples, validate on 6591 samples
Epoch 1/10
26359/26359 [==============================] - 2s 65us/sample - loss: 0.4118 - accuracy: 0.8465 - val_loss: 0.3154 - val_accuracy: 0.8935
Epoch 2/10
26359/26359 [==============================] - 1s 24us/sample - loss: 0.3181 - accuracy: 0.8922 - val_loss: 0.3029 - val_accuracy: 0.8952
Epoch 3/10
26359/26359 [==============================] - 1s 24us/sample - loss: 0.3057 - accuracy: 0.8935 - val_loss: 0.2961 - val_accuracy: 0.8944
Epoch 4/10
26359/26359 [==============================] - 1s 24us/sample - loss: 0.2995 - accuracy: 0.8951 - val_loss: 0.2930 - val_accuracy: 0.8933
Epoch 5/10
26359/26359 [==============================] - 1s 26us/sample - loss: 0.2948 - accuracy: 0.8957 - val_loss: 0.2915 - val_accuracy: 0.8941
Epoch 6/10
26359/26359 [==============================] - 1s 26us/sample - loss: 0.2904 - accuracy: 0.8964 - val_loss: 0.2901 - val_accuracy: 0.8935
Epoch 7/10
26359/26359 [==============================] - 1s 24us/sample - loss: 0.2881 - accuracy: 0.8968 - val_loss: 0.2894 - val_accuracy: 0.8932
Epoch 8/10
26359/26359 [==============================] - 1s 24us/sample - loss: 0.2859 - accuracy: 0.8954 - val_loss: 0.2879 - val_accuracy: 0.8936
Epoch 9/10
26359/26359 [==============================] - 1s 24us/sample - loss: 0.2843 - accuracy: 0.8973 - val_loss: 0.2879 - val_accuracy: 0.8956
Epoch 10/10
26359/26359 [==============================] - 1s 24us/sample - loss: 0.2838 - accuracy: 0.8981 - val_loss: 0.2865 - val_accuracy: 0.8949
```

```
# 損失
plt.figure()
plt.plot(history.epoch, history.history['loss'], label='Train Loss')
plt.plot(history.epoch, history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# 正解率
plt.figure()
plt.plot(history.epoch, history.history['accuracy'], label='Train Accuracy')
plt.plot(history.epoch, history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





テストデータを使って、精度を調べます。

```
loss, acc = model.evaluate(x_test, y_test)
print('損失：', loss, '正解率：', acc)
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class 'pandas.core.frame.Dat
8238/8238 [==============================] - 0s 46us/sample - loss: 0.2798 - accuracy: 0.8992
損失： 0.2798066571532132 正解率： 0.8992474
```

推論

推論は `model.predict()` で行います。
戻り値はリストになっていて、内容は `[ラベル0である確率, ラベル1である確率]` です。
今回はラベル 0 は no、ラベル1は yes ですので、 `[no である確率, yes である確率]` と読み替えます。
出力層の活性化関数は「softmax」ですので、２つの確率の合計は 1 になります。

```
y_pred = model.predict(x_test)
y_pred[:10]
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class 'pandas.core.frame.Dat
```

```
array([[0.9412501 , 0.05874996],
       [0.8887464 , 0.11125363],
       [0.88974166, 0.11025836],
       [0.9470079 , 0.05299216],
       [0.9754937 , 0.02450626],
       [0.66857594, 0.3314241 ],
       [0.5096008 , 0.4903992 ],
       [0.95786005, 0.04213991],
       [0.9620996 , 0.0379004 ],
       [0.9265177 , 0.07348234]], dtype=float32)
```

それぞれの確立の値をどう解釈するかは人間が決めます。
ここでは、yesの確率が50%以上だった場合に、yesと分類することにします。

正解のラベルと比較してみましょう。

```
def decision_y(probability_list):

  # yesの確立が50%以上なら'yes'として分類する
  return 'yes' if probability_list[1] >= 0.5 else 'no'
```

```
label_names = {0:'no', 1:'yes'}

for i, pred in enumerate(y_pred[:20]):

  # 予測値。yes の確率が50%以上ならyesとして扱う
  pred_yes_no = decision_y(pred)

  # 正解のラベル
  true_yes_no = y_test.iloc[i].idxmax()

  result = ''

  if true_yes_no == 'yes' and pred_yes_no == 'yes':
    result = 'True Positive'
  elif true_yes_no == 'yes' and pred_yes_no == 'no':
    result = 'False Negative'
  elif true_yes_no == 'no' and pred_yes_no == 'yes':
    result = 'True Negative'
  elif true_yes_no == 'no' and pred_yes_no == 'no':
    result = 'False Positive'

  print(f'yesの確率：{pred[1]:>6.2%}  予測：{pred_yes_no:3}  正解：{true_yes_no:3}  {result}')
```

```
yesの確率：  5.87%  予測：no   正解：no   False Positive
yesの確率：11.13%  予測：no   正解：no   False Positive
yesの確率：11.03%  予測：no   正解：no   False Positive
yesの確率：  5.30%  予測：no   正解：no   False Positive
yesの確率：  2.45%  予測：no   正解：no   False Positive
yesの確率：33.14%  予測：no   正解：no   False Positive
yesの確率：49.04%  予測：no   正解：no   False Positive
yesの確率：  4.21%  予測：no   正解：no   False Positive
yesの確率：  3.79%  予測：no   正解：no   False Positive
yesの確率：  7.35%  予測：no   正解：no   False Positive
yesの確率：  3.51%  予測：no   正解：no   False Positive
yesの確率：  7.08%  予測：no   正解：no   False Positive
yesの確率：26.52%  予測：no   正解：no   False Positive
yesの確率：35.40%  予測：no   正解：no   False Positive
yesの確率：  3.45%  予測：no   正解：no   False Positive
yesの確率：  9.48%  予測：no   正解：no   False Positive
yesの確率：16.97%  予測：no   正解：no   False Positive
yesの確率：11.96%  予測：no   正解：no   False Positive
yesの確率：  9.32%  予測：no   正解：no   False Positive
yesの確率：45.10%  予測：no   正解：yes  False Negative
```

評価

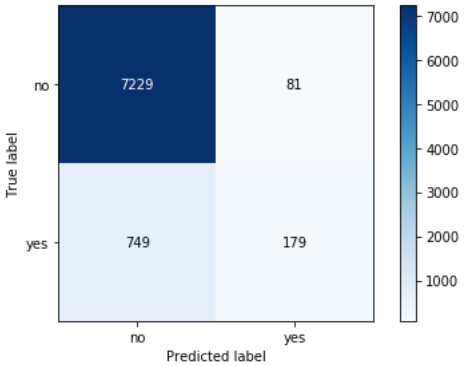
二値分類の予測結果は、以下のように分けることができます。

- True Positive（TP、真陽性） … 真の値を真と予測した
- False Negative（FN、偽陰性） … 真の値を偽と予測した
- False Positive(FP、偽陽性) … 偽の値を真と予測した
- True Negative（TN、真陰性） … 偽の値を偽と予測した

		正解	
		真	偽
予測	真	True Positive (TP)	False Positive (FP)
	偽	False Negative (FN)	True Negative (TN)

今回の予測結果を、混同行列で表示してみましょう。
`sklearn.metrics.confusion_matrix()` を使うと、混同行列を表示できます。

```
#  ラベルと予測を yes / no のリストに変換
<p class="mume-header " id="ラベルと予測を-yes-no-のリストに変換"></p>

y_test_yes_no = y_test.idxmax(axis=1)
y_pred_yes_no = [decision_y(i) for i in y_pred]


#  混同行列を描画する関数
<p class="mume-header " id="混同行列を描画する関数"></p>

def plot_confusion_matrix(test_list, pred_list):

  # 混同行列を作成
  cm = confusion_matrix(test_list, pred_list)

  classes = ['no', 'yes']

  fig, ax = plt.subplots()
  im = ax.imshow(cm, interpolation='nearest', cmap='Blues')
  ax.figure.colorbar(im, ax=ax)
  ax.set(xticks=np.arange(cm.shape[1]),
         yticks=np.arange(cm.shape[0]),
         xticklabels=classes, yticklabels=classes,  # クラス名を表示
         ylabel='True label',
         xlabel='Predicted label')

  thresh = cm.max() / 2.
  for i in range(cm.shape[0]):
      for j in range(cm.shape[1]):
          ax.text(j, i, cm[i, j],
                  ha="center", va="center",
                  color="white" if cm[i, j] > thresh else "black")

  fig.tight_layout()

  return ax


#  混同行列を描画
<p class="mume-header " id="混同行列を描画"></p>

plot_confusion_matrix(y_test_yes_no, y_pred_yes_no)


<matplotlib.axes._subplots.AxesSubplot at 0x7f3c0e670b38>
```



TP、FP、FN、TNの値から、以下の指標を算出します。

- Accuracy（正解率） … 予測全体のうち、正しく予測できた割合。

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- Precision（適合率）… 真と予測したうち、実際に真だった割合。FPを低く抑えたい場合に採用する。

$$Precision = \frac{TP}{TP + FP}$$

- Recall（再現率）… 実際に真のうち、真と予測できた割合。FNを低く抑えたい場合に採用する。

$$Recall = \frac{TP}{TP + FN}$$

- Specificity（特異率）…実際に偽のうち、偽と予測できた割合。

$$Specificity = \frac{TN}{FP + TN}$$

- F-score（F値） … Precision（適合率） と Recall（再現率） の調和平均。0 が最も悪く、1 が最も良い。

$$Fscore = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision}$$

上の指標を多クラス分類に当てはめると、以下のように言い換えることができます。

- 全体の指標
  - Accuracy（正解率）
- 個々のクラスの指標
  - Precision（適合率）…そのクラスと予想したうち、実際にそのクラスだった割合。
  - Recall（再現率）…実際にそのクラスのうち、そのクラスと予測できた割合。
  - F-socre（F値）

上記の指標は、 `sklearn.metrics.classification_report()` を使うと一度に調べることができます。

```
#  指標の表示
<p class="mume-header " id="指標の表示"></p>

print(classification_report(y_test_yes_no, y_pred_yes_no))
```

```
              precision    recall  f1-score   support

          no       0.91      0.99      0.95      7310
         yes       0.69      0.19      0.30       928

    accuracy                           0.90      8238
   macro avg       0.80      0.59      0.62      8238
weighted avg       0.88      0.90      0.87      8238
```

Accuracy（正解率）は約 90 % でした。
一見すると良い値に思えますが、今回のデータセットには、「yes」のデータは 1 割しか含まれていなかったことを思い出してください。
つまり、すべて「no」と予測しても、正解率は 90 % 程度になりますが、それでは役に立ちません。

今回のようなケースでは、「見込み客」を「見込みなし」に誤分類してしまい、セールスチャンスを逃してしまうことを防ぎたいため、「yes」の Recall（再現率）の値に注目します。
今回作成したモデルでは、「yes」のRecall は 約 20% 、つまり残り 80 % の見込み客を見逃しています。

不均衡データの解消

クラスのデータ数が均等でないデータを、不均衡データと呼びます。
不均衡データの少数派データの予測精度を上げるには、以下の方法などで工夫します。

- アンダーサンプリング … 多数派のデータを減らす
- オーバーサンプリング … 少数派のデータを水増しする
- アンダーサンプリングとオーバーサンプリングの両方を行う
- 損失関数をカスタマイズして、誤分類に対するペナルティを重くする

ここではサンプリングを行って、精度が向上するか試してみましょう。

アンダーサンプリング

アンダーサンプリングは、多数派のデータをサンプリングして（減らして）バランスをとる手法です。
ランダムにサンプリングすると、データのパターンに偏りが発生する場合があります。
そこで、多数派の中でデータのクラスタリングを行い、それぞれのクラスタから偏りがないようにサンプリングする手法がよく用いられます。

アンダーサンプリング、オーバーサンプリングは、imbalanced-learnというライブラリで簡単に行うことできます。

```
#  データとラベルに分ける
<p class="mume-header " id="データとラベルに分ける-1"></p>

x = df_wk.drop('y', axis=1).values
y = df_wk['y'].values

#  元の yes のデータ数
<p class="mume-header " id="元の-yes-のデータ数"></p>

yes_sample_count = df_wk[df_wk['y'] == 'yes']['age'].count()
#  元の no のデータ数
<p class="mume-header " id="元の-no-のデータ数"></p>

no_sample_count = df_wk[df_wk['y'] == 'no']['age'].count()

print('元のyesの件数', yes_sample_count)
print('元のnoの件数', no_sample_count)
```

```
元のyesの件数 4640
元のnoの件数 36548
```

```
from imblearn.under_sampling import EditedNearestNeighbours

#  アンダーサンプリング
<p class="mume-header " id="アンダーサンプリング"></p>

enn = EditedNearestNeighbours(random_state=777)
x_resampled, y_resampled = enn.fit_resample(x, y)

print('サンプル後 yes:', np.count_nonzero(y_resampled=='yes'))
print('サンプル後  no:', np.count_nonzero(y_resampled=='no'))

#  訓練データ8割、評価データ2割に分割
<p class="mume-header " id="訓練データ8割-評価データ2割に分割"></p>

x_train, x_test, y_train, y_test = train_test_split(x_resampled, y_resampled, test_size=0.2, random_state=777)
#  更に訓練データ8割、検証データ2割に分割
<p class="mume-header " id="更に訓練データ8割-検証データ2割に分割"></p>

x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=777)

x_train = pd.DataFrame(x_train)
y_train = pd.get_dummies(y_train)
x_val = pd.DataFrame(x_val)
y_val = pd.get_dummies(y_val)
x_test = pd.DataFrame(x_test)
y_test = pd.get_dummies(y_test)
```

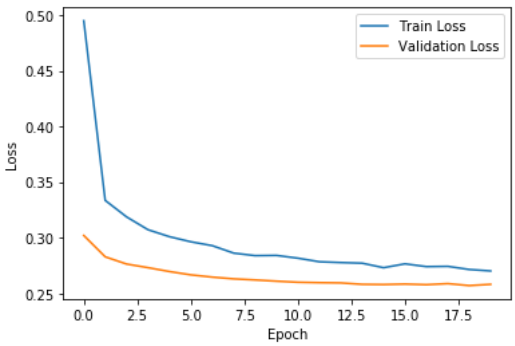
```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: DeprecationWarning: The module is deprecated in version 0.21 and will be re
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

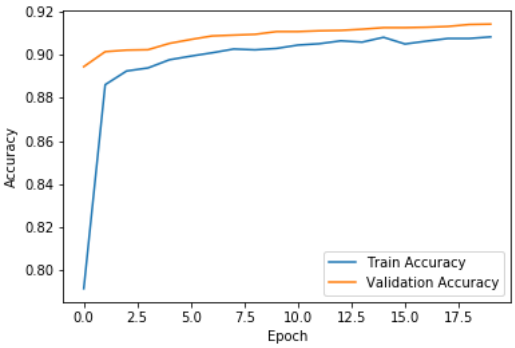
```
サンプル後 yes: 4640
サンプル後  no: 29436
```

```
#  標準化
<p class="mume-header " id="標準化-1"></p>


#  各列の平均
<p class="mume-header " id="各列の平均"></p>

mean = x_train.mean()
#  各列の標準偏差
<p class="mume-header " id="各列の標準偏差"></p>

std = x_train.std(ddof=False)

#  平均を引き、標準偏差で割る
<p class="mume-header " id="平均を引き-標準偏差で割る"></p>

x_train -= mean
x_train /= std

x_train.describe()

#  検証データ、評価データも同様に処理する
<p class="mume-header " id="検証データ-評価データも同様に処理する"></p>

x_val -= mean
x_val /= std

x_test -= mean
x_test /= std
```

```
#  モデルの定義
<p class="mume-header " id="モデルの定義-1"></p>

model = tf.keras.Sequential([
  layers.Dense(16, activation='relu', input_shape=(len(x_train.columns), )),
  layers.Dropout(0.2),
  layers.Dense(16, activation='relu'),
  layers.Dropout(0.1),
  layers.Dense(2, activation='softmax')
])

#  モデルのコンパイル
<p class="mume-header " id="モデルのコンパイル"></p>

model.compile(optimizer='adam',                      # 最適化アルゴリズム…Adam
              loss='categorical_crossentropy',       # 損失関数…Categorical Crossentropy
              metrics=['accuracy'])                  # 評価関数…正解率

#  学習とエポックごとの評価
<p class="mume-header " id="学習とエポックごとの評価"></p>

history = model.fit(x_train,                           # 訓練データ
                    y_train,                           # 訓練データのラベル
                    batch_size=128,                    # バッチサイズ
                    epochs=20,                         # エポック数
                    validation_data=(x_val, y_val))    # 検証データ

#  損失
<p class="mume-header " id="損失"></p>

plt.figure()
plt.plot(history.epoch, history.history['loss'], label='Train Loss')
plt.plot(history.epoch, history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

#  正解率
<p class="mume-header " id="正解率"></p>

plt.figure()
plt.plot(history.epoch, history.history['accuracy'], label='Train Accuracy')
plt.plot(history.epoch, history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

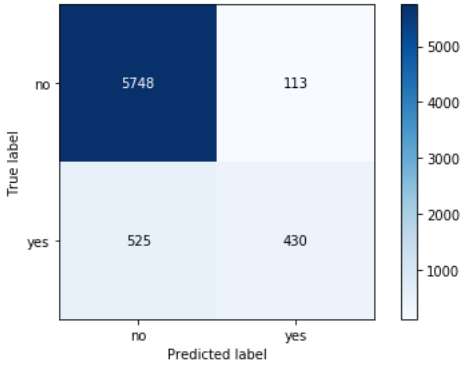
```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class 'pandas.core.frame.Dat
Train on 21808 samples, validate on 5452 samples
Epoch 1/20
21808/21808 [==============================] - 1s 32us/sample - loss: 0.4948 - accuracy: 0.7913 - val_loss: 0.3020 - val_accuracy: 0.8945
Epoch 2/20
21808/21808 [==============================] - 1s 25us/sample - loss: 0.3336 - accuracy: 0.8861 - val_loss: 0.2829 - val_accuracy: 0.9015
Epoch 3/20
21808/21808 [==============================] - 1s 26us/sample - loss: 0.3187 - accuracy: 0.8925 - val_loss: 0.2764 - val_accuracy: 0.9022
Epoch 4/20
21808/21808 [==============================] - 1s 24us/sample - loss: 0.3072 - accuracy: 0.8939 - val_loss: 0.2731 - val_accuracy: 0.9024
Epoch 5/20
21808/21808 [==============================] - 1s 26us/sample - loss: 0.3010 - accuracy: 0.8977 - val_loss: 0.2697 - val_accuracy: 0.9054
Epoch 6/20
21808/21808 [==============================] - 1s 25us/sample - loss: 0.2964 - accuracy: 0.8994 - val_loss: 0.2667 - val_accuracy: 0.9072
Epoch 7/20
21808/21808 [==============================] - 1s 25us/sample - loss: 0.2929 - accuracy: 0.9010 - val_loss: 0.2647 - val_accuracy: 0.9088
Epoch 8/20
21808/21808 [==============================] - 1s 24us/sample - loss: 0.2862 - accuracy: 0.9027 - val_loss: 0.2631 - val_accuracy: 0.9092
Epoch 9/20
21808/21808 [==============================] - 1s 24us/sample - loss: 0.2839 - accuracy: 0.9024 - val_loss: 0.2621 - val_accuracy: 0.9096
Epoch 10/20
21808/21808 [==============================] - 1s 26us/sample - loss: 0.2841 - accuracy: 0.9030 - val_loss: 0.2610 - val_accuracy: 0.9109
Epoch 11/20
21808/21808 [==============================] - 1s 24us/sample - loss: 0.2817 - accuracy: 0.9046 - val_loss: 0.2600 - val_accuracy: 0.9109
Epoch 12/20
21808/21808 [==============================] - 1s 25us/sample - loss: 0.2784 - accuracy: 0.9052 - val_loss: 0.2597 - val_accuracy: 0.9112
Epoch 13/20
21808/21808 [==============================] - 1s 25us/sample - loss: 0.2777 - accuracy: 0.9065 - val_loss: 0.2594 - val_accuracy: 0.9114
Epoch 14/20
21808/21808 [==============================] - 1s 26us/sample - loss: 0.2772 - accuracy: 0.9060 - val_loss: 0.2582 - val_accuracy: 0.9120
Epoch 15/20
21808/21808 [==============================] - 1s 26us/sample - loss: 0.2731 - accuracy: 0.9082 - val_loss: 0.2580 - val_accuracy: 0.9127
Epoch 16/20
21808/21808 [==============================] - 1s 24us/sample - loss: 0.2766 - accuracy: 0.9051 - val_loss: 0.2584 - val_accuracy: 0.9127
Epoch 17/20
21808/21808 [==============================] - 1s 24us/sample - loss: 0.2741 - accuracy: 0.9064 - val_loss: 0.2579 - val_accuracy: 0.9129
Epoch 18/20
21808/21808 [==============================] - 1s 24us/sample - loss: 0.2743 - accuracy: 0.9076 - val_loss: 0.2588 - val_accuracy: 0.9132
Epoch 19/20
21808/21808 [==============================] - 1s 25us/sample - loss: 0.2715 - accuracy: 0.9076 - val_loss: 0.2570 - val_accuracy: 0.9142
Epoch 20/20
21808/21808 [==============================] - 1s 24us/sample - loss: 0.2702 - accuracy: 0.9084 - val_loss: 0.2582 - val_accuracy: 0.9143
```





```
#  推論
<p class="mume-header " id="推論-1"></p>

y_pred = model.predict(x_test)

#  ラベルと予測を yes / no のリストに変換
<p class="mume-header " id="ラベルと予測を-yes-no-のリストに変換-1"></p>

y_test_yes_no = y_test.idxmax(axis=1)
y_pred_yes_no = [decision_y(i) for i in y_pred]

#  混同行列を描画
<p class="mume-header " id="混同行列を描画-1"></p>

plot_confusion_matrix(y_test_yes_no, y_pred_yes_no)

#  指標の表示
<p class="mume-header " id="指標の表示-1"></p>

print(classification_report(y_test_yes_no, y_pred_yes_no))
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class 'pandas.core.frame.Dat
              precision    recall  f1-score   support

          no       0.92      0.98      0.95      5861
         yes       0.79      0.45      0.57       955

    accuracy                           0.91      6816
   macro avg       0.85      0.72      0.76      6816
weighted avg       0.90      0.91      0.90      6816
```



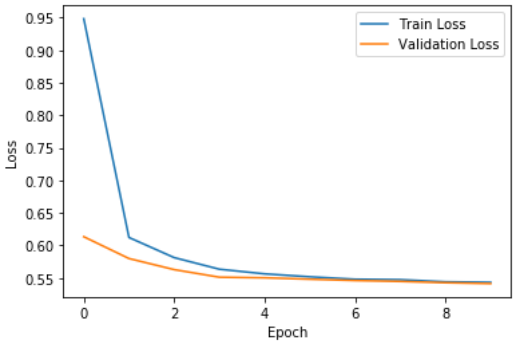
yes の再現率（Recall）は約50%まで向上しました。

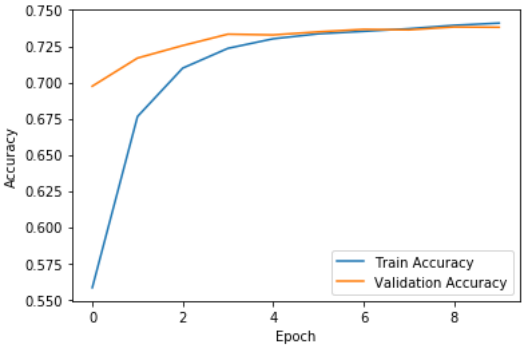
オーバーサンプリング

オーバーサンプリングは、少数派のデータを水増し（拡張）して件数を増やす手法です。
元のデータをコピーし、ノイズを加えて水増しします。

```
from imblearn.over_sampling import SMOTE

#  オーバーサンプリング
<p class="mume-header " id="オーバーサンプリング"></p>

smote = SMOTE(random_state=777)
x_resampled, y_resampled = smote.fit_resample(x, y)

print('サンプル後 yes:', np.count_nonzero(y_resampled=='yes'))
print('サンプル後  no:', np.count_nonzero(y_resampled=='no'))

#  訓練データ8割、評価データ2割に分割
<p class="mume-header " id="訓練データ8割-評価データ2割に分割-1"></p>

x_train, x_test, y_train, y_test = train_test_split(x_resampled, y_resampled, test_size=0.2, random_state=777)
#  更に訓練データ8割、検証データ2割に分割
<p class="mume-header " id="更に訓練データ8割-検証データ2割に分割-1"></p>

x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=777)

x_train = pd.DataFrame(x_train)
y_train = pd.get_dummies(y_train)
x_val = pd.DataFrame(x_val)
y_val = pd.get_dummies(y_val)
x_test = pd.DataFrame(x_test)
y_test = pd.get_dummies(y_test)


サンプル後 yes: 36548
サンプル後  no: 36548


#  モデルの定義
<p class="mume-header " id="モデルの定義-2"></p>

model = tf.keras.Sequential([
  layers.Dense(16, activation='relu', input_shape=(len(x_train.columns), )),
  layers.Dropout(0.2),
  layers.Dense(16, activation='relu'),
  layers.Dropout(0.1),
  layers.Dense(2, activation='softmax')
])

#  モデルのコンパイル
<p class="mume-header " id="モデルのコンパイル-1"></p>

model.compile(optimizer='adam',                      # 最適化アルゴリズム…Adam
              loss='categorical_crossentropy',       # 損失関数…Categorical Crossentropy
              metrics=['accuracy'])                  # 評価関数…正解率

#  学習とエポックごとの評価
<p class="mume-header " id="学習とエポックごとの評価-1"></p>

history = model.fit(x_train,                           # 訓練データ
                    y_train,                           # 訓練データのラベル
                    batch_size=128,                    # バッチサイズ
                    epochs=10,                         # エポック数
                    validation_data=(x_val, y_val))    # 検証データ

#  損失
<p class="mume-header " id="損失-1"></p>

plt.figure()
plt.plot(history.epoch, history.history['loss'], label='Train Loss')
plt.plot(history.epoch, history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

#  正解率
<p class="mume-header " id="正解率-1"></p>

plt.figure()
plt.plot(history.epoch, history.history['accuracy'], label='Train Accuracy')
plt.plot(history.epoch, history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

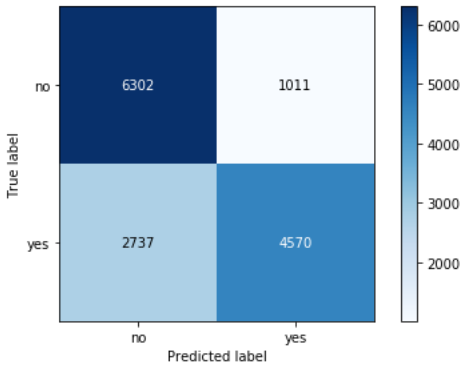
```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class 'pandas.core.frame.Dat
Train on 46780 samples, validate on 11696 samples
Epoch 1/10
46780/46780 [==============================] - 1s 27us/sample - loss: 0.9482 - accuracy: 0.5583 - val_loss: 0.6134 - val_accuracy: 0.6974
Epoch 2/10
46780/46780 [==============================] - 1s 24us/sample - loss: 0.6122 - accuracy: 0.6764 - val_loss: 0.5798 - val_accuracy: 0.7168
Epoch 3/10
46780/46780 [==============================] - 1s 23us/sample - loss: 0.5813 - accuracy: 0.7099 - val_loss: 0.5628 - val_accuracy: 0.7255
Epoch 4/10
46780/46780 [==============================] - 1s 24us/sample - loss: 0.5634 - accuracy: 0.7235 - val_loss: 0.5511 - val_accuracy: 0.7333
Epoch 5/10
46780/46780 [==============================] - 1s 24us/sample - loss: 0.5563 - accuracy: 0.7301 - val_loss: 0.5502 - val_accuracy: 0.7328
Epoch 6/10
46780/46780 [==============================] - 1s 24us/sample - loss: 0.5518 - accuracy: 0.7336 - val_loss: 0.5481 - val_accuracy: 0.7350
Epoch 7/10
46780/46780 [==============================] - 1s 24us/sample - loss: 0.5480 - accuracy: 0.7353 - val_loss: 0.5460 - val_accuracy: 0.7367
Epoch 8/10
46780/46780 [==============================] - 1s 23us/sample - loss: 0.5474 - accuracy: 0.7371 - val_loss: 0.5447 - val_accuracy: 0.7363
Epoch 9/10
46780/46780 [==============================] - 1s 23us/sample - loss: 0.5441 - accuracy: 0.7394 - val_loss: 0.5427 - val_accuracy: 0.7383
Epoch 10/10
46780/46780 [==============================] - 1s 24us/sample - loss: 0.5433 - accuracy: 0.7410 - val_loss: 0.5414 - val_accuracy: 0.7381
```






```
#  推論
<p class="mume-header " id="推論-2"></p>

y_pred = model.predict(x_test)

#  ラベルと予測を yes / no のリストに変換
<p class="mume-header " id="ラベルと予測を-yes-no-のリストに変換-2"></p>

y_test_yes_no = y_test.idxmax(axis=1)
y_pred_yes_no = [decision_y(i) for i in y_pred]

#  混同行列を描画
<p class="mume-header " id="混同行列を描画-2"></p>

plot_confusion_matrix(y_test_yes_no, y_pred_yes_no)

#  指標の表示
<p class="mume-header " id="指標の表示-2"></p>

print(classification_report(y_test_yes_no, y_pred_yes_no))
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class 'pandas.core.frame.Dat
              precision    recall  f1-score   support

          no       0.70      0.86      0.77      7313
         yes       0.82      0.63      0.71      7307

    accuracy                           0.74     14620
   macro avg       0.76      0.74      0.74     14620
weighted avg       0.76      0.74      0.74     14620
```



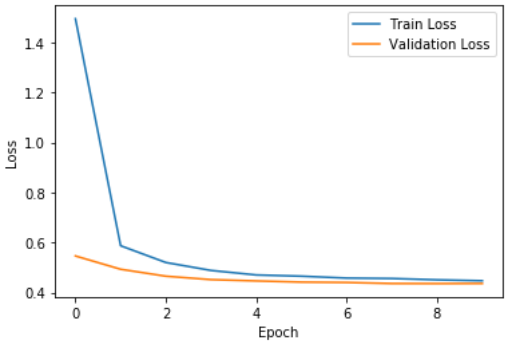
yesの再現率（Recall）は約 60 %まで向上しました。

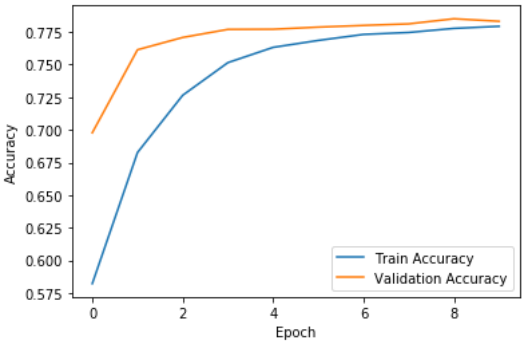
アンダーサンプリング&オーバーサンプリング

アンダーサンプリングとオーバーサンプリングの両方を行ってみましょう。

```
from imblearn.combine import SMOTEENN


smoteenn = SMOTEENN(random_state=777)
x_resampled, y_resampled = smoteenn.fit_resample(x, y)

print('サンプル後 yes:', np.count_nonzero(y_resampled=='yes'))
print('サンプル後  no:', np.count_nonzero(y_resampled=='no'))

#  訓練データ8割、評価データ2割に分割
<p class="mume-header " id="訓練データ8割-評価データ2割に分割-2"></p>

x_train, x_test, y_train, y_test = train_test_split(x_resampled, y_resampled, test_size=0.2, random_state=777)
#  更に訓練データ8割、検証データ2割に分割

<p class="mume-header " id="更に訓練データ8割-検証データ2割に分割-2"></p>

x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=777)

x_train = pd.DataFrame(x_train)
y_train = pd.get_dummies(y_train)
x_val = pd.DataFrame(x_val)
y_val = pd.get_dummies(y_val)
x_test = pd.DataFrame(x_test)
y_test = pd.get_dummies(y_test)


サンプル後 yes: 33478
サンプル後  no: 24178
```

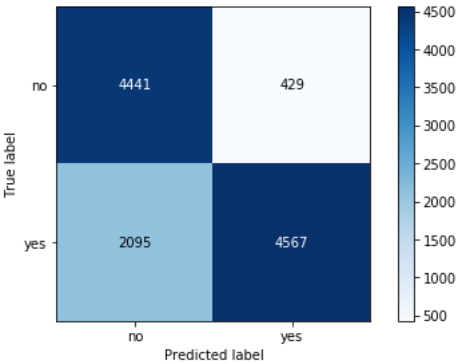
```
#  モデルの定義
<p class="mume-header " id="モデルの定義-3"></p>

model = tf.keras.Sequential([
  layers.Dense(16, activation='relu', input_shape=(len(x_train.columns), )),
  layers.Dropout(0.2),
  layers.Dense(16, activation='relu'),
  layers.Dropout(0.1),
  layers.Dense(2, activation='softmax')
])

#  モデルのコンパイル
<p class="mume-header " id="モデルのコンパイル-2"></p>

model.compile(optimizer='adam',                      # 最適化アルゴリズム…Adam
              loss='categorical_crossentropy',       # 損失関数…Categorical Crossentropy
              metrics=['accuracy'])                  # 評価関数…正解率

#  学習とエポックごとの評価
<p class="mume-header " id="学習とエポックごとの評価-2"></p>

history = model.fit(x_train,                           # 訓練データ
                    y_train,                           # 訓練データのラベル
                    batch_size=128,                    # バッチサイズ
                    epochs=10,                         # エポック数
                    validation_data=(x_val, y_val))    # 検証データ

#  損失
<p class="mume-header " id="損失-2"></p>

plt.figure()
plt.plot(history.epoch, history.history['loss'], label='Train Loss')
plt.plot(history.epoch, history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

#  正解率
<p class="mume-header " id="正解率-2"></p>

plt.figure()
plt.plot(history.epoch, history.history['accuracy'], label='Train Accuracy')
plt.plot(history.epoch, history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()


WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class 'pandas.core.frame.Dat
Train on 36899 samples, validate on 9225 samples
Epoch 1/10
36899/36899 [==============================] - 1s 28us/sample - loss: 1.4948 - accuracy: 0.5823 - val_loss: 0.5460 - val_accuracy: 0.6978
Epoch 2/10
36899/36899 [==============================] - 1s 25us/sample - loss: 0.5869 - accuracy: 0.6826 - val_loss: 0.4928 - val_accuracy: 0.7613
Epoch 3/10
36899/36899 [==============================] - 1s 27us/sample - loss: 0.5199 - accuracy: 0.7265 - val_loss: 0.4650 - val_accuracy: 0.7707
Epoch 4/10
36899/36899 [==============================] - 1s 25us/sample - loss: 0.4881 - accuracy: 0.7515 - val_loss: 0.4513 - val_accuracy: 0.7769
Epoch 5/10
36899/36899 [==============================] - 1s 24us/sample - loss: 0.4700 - accuracy: 0.7631 - val_loss: 0.4461 - val_accuracy: 0.7770
Epoch 6/10
36899/36899 [==============================] - 1s 24us/sample - loss: 0.4651 - accuracy: 0.7684 - val_loss: 0.4414 - val_accuracy: 0.7786
Epoch 7/10
36899/36899 [==============================] - 1s 24us/sample - loss: 0.4575 - accuracy: 0.7731 - val_loss: 0.4399 - val_accuracy: 0.7799
Epoch 8/10
36899/36899 [==============================] - 1s 24us/sample - loss: 0.4559 - accuracy: 0.7745 - val_loss: 0.4357 - val_accuracy: 0.7811
Epoch 9/10
36899/36899 [==============================] - 1s 25us/sample - loss: 0.4505 - accuracy: 0.7777 - val_loss: 0.4355 - val_accuracy: 0.7850
Epoch 10/10
36899/36899 [==============================] - 1s 24us/sample - loss: 0.4468 - accuracy: 0.7793 - val_loss: 0.4358 - val_accuracy: 0.7831
```






```
#  推論
<p class="mume-header " id="推論-3"></p>

y_pred = model.predict(x_test)

#  ラベルと予測を yes / no のリストに変換
<p class="mume-header " id="ラベルと予測を-yes-no-のリストに変換-3"></p>

y_test_yes_no = y_test.idxmax(axis=1)
y_pred_yes_no = [decision_y(i) for i in y_pred]

#  混同行列を描画
<p class="mume-header " id="混同行列を描画-3"></p>

plot_confusion_matrix(y_test_yes_no, y_pred_yes_no)

#  指標の表示
<p class="mume-header " id="指標の表示-3"></p>

print(classification_report(y_test_yes_no, y_pred_yes_no))
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class 'pandas.core.frame.Dat
              precision    recall  f1-score   support

          no       0.68      0.91      0.78      4870
         yes       0.91      0.69      0.78      6662

    accuracy                           0.78     11532
   macro avg       0.80      0.80      0.78     11532
weighted avg       0.82      0.78      0.78     11532
```



yes の再現率（Recall）は約70%まで向上しました。

アルゴリズムによる分類（ランダムフォレスト）

ここまでニューラルネットワークで分類に挑戦してきましたが、アルゴリズムによる分類も試してみましょう。
ランダムフォレストで分類したらどうなるでしょうか。

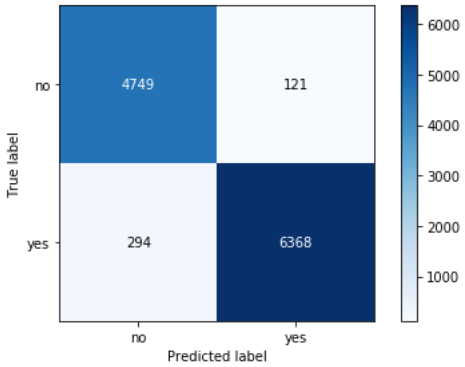
```
from sklearn.ensemble import RandomForestClassifier

#  ランダムフォレストのモデル
<p class="mume-header " id="ランダムフォレストのモデル"></p>

random_forest = RandomForestClassifier(n_estimators=100)

#  学習
<p class="mume-header " id="学習-1"></p>

random_forest.fit(x_train, y_train)

#  推論
<p class="mume-header " id="推論-4"></p>

y_pred = random_forest.predict(x_test)
y_pred
```

```
array([[0, 1],
       [0, 1],
       [1, 0],
       ...,
       [1, 0],
       [0, 1],
       [1, 0]], dtype=uint8)
```

```
#  リストを整形
<p class="mume-header " id="リストを整形"></p>

y_test_yes_no = y_test.values.argmax(axis=1)
y_pred_yes_no = y_pred.argmax(axis=1)

#  指標の表示
<p class="mume-header " id="指標の表示-4"></p>

print(classification_report(y_test_yes_no, y_pred_yes_no))

#  正解と推論結果の混同行列を描画
<p class="mume-header " id="正解と推論結果の混同行列を描画"></p>

plot_confusion_matrix(y_test_yes_no, y_pred_yes_no)
```

```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96      4870
           1       0.98      0.96      0.97      6662

    accuracy                           0.96     11532
   macro avg       0.96      0.97      0.96     11532
weighted avg       0.96      0.96      0.96     11532
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c05da4630>
```



ニューラルネットワークモデルよりも、ランダムフォレストモデルの方が良い結果になりました。
正解率（Accurach）が約 96 %と高く、yes と no 両方のF1-scoreのバランスも取れています。

必ずしもニューラルネットワークが一番良い結果になるとは限らないため、色々なモデルを試してみると良いでしょう。

まとめ

ここまで、以下のことを学習しました。

- 回帰と分類の違い
  - 予測する対象の違い
    回帰 … 値（金額、気温など）を 1 つ予測します。
    分類 … どのクラス（1…犬、 2…ネコなど）に属するかを予測します。
  - モデルの出力の違い
    回帰 … モデルの出力をそのまま出力します。出力数は1で、活性化関数は使用しません。
    分類 … それぞれのクラスに属する確率を出力します。出力数はクラスの数で、活性化関数はsoftmaxを使います。
- one-hotベクトル
  one-hotベクトルとは、該当するものだけ1、その他が0のリストです。
  カテゴリ変数（金額のように大きさ・量を表す数値ではなく、1…犬、2…ネコのような意味をもつ値）は、one-hotベクトルに変換して学習します。
- 分類モデルの評価方法
  2値分類の予測結果は、以下のように分けることができます。
  - True Positive（TP、真陽性） … 正の値を正と予測した
  - False Negative（FN、偽陰性） … 正の値を負と予測した
  - False Positive(FP、偽陽性) … 負の値を正と予測した
  - True Negative（TN、真陰性） … 負の値を負と予測した

  予測結果から、以下の指標が算出できます。
  ビジネスの目的に合わせてどの指標に注目するかを決めます。
  - Accuracy（正解率） … 予測全体のうち、正しく予測できた割合。

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

  - Precision（適合率）… 正と予測したうち、実際に正だった割合。FPを低く抑えたい場合に採用する。

$$Precision = \frac{TP}{TP + FP}$$

  - Recall（再現率）… 実際に正のうち、正と予測できた割合。FNを低く抑えたい場合に採用する。

$$Recall = \frac{TP}{TP + FN}$$

  - Specificity（特異率）…実際に負のうち、負と予測できた割合。Recall（再現率）の逆。

$$Specificity = \frac{TN}{FP + TN}$$

  - F-score（F値） … Precision と Recall の調和平均。0 が最も悪く、1 が最も良い。

$$Fscore = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision}$$

- 不均衡データの取り扱い
  分類するクラス間のデータ数に差があることを、不均衡データと呼びます。

不均衡データでは、すべて多数派のデータに分類するだけで高い正解率となってしまうため、他の指標にも注目する必要があります。
少数派のクラスはうまく学習できない傾向にあるため、サンプリングを行ってデータ数のバランスをとることも有効です。

- アンダーサンプリング … 多数派のデータを減らします。
- オーバーサンプリング … 少数派のデータを水増しします。

AI2-3

AIの実装（教師なし学習）

－ 講義内容 －

- 教師なし学習の概要
- クラスタリング
- 演習：クラスタリング
- 次元削減
- 演習：次元削減

教師なし学習
（クラスタリング・次元削減）

2019年11月26日（火）

新潟人工知能研究所 技術開発部

大目次

p03. 科目概要

p06. クラスタリング

p13. 次元削減

科目概要

株式会社新潟人工知能研究所
技術開発部

シラバス

- 教師なし学習の一種である「クラスタリング」の概要を理解する。特に教師あり学習における「分類」モデルとの違いを理解する。
- また、クラスタリングの代表的なアルゴリズムである k 平均法（k-means）の理論を理解して、Pythonによる実装を行う。
- さらに、教師なし学習の「主成分分析（PCA）」の理論を理解して、Pythonによる次元削減の実装が適切に行えるようにする。

教師なし学習の概要

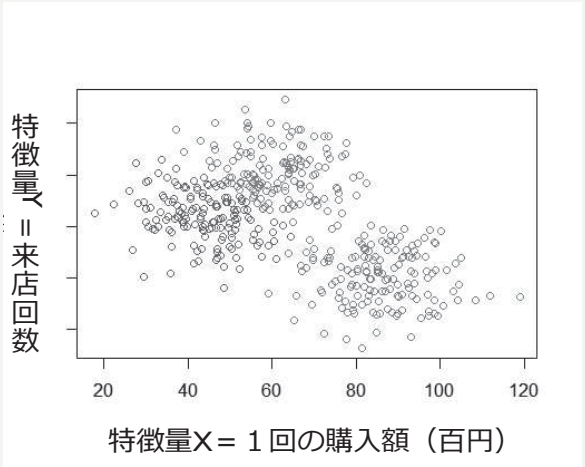
	教師あり学習	教師なし学習	強化学習
学習方法	正解（目的変数）がある訓練データを大量に分析して、パターンを予測するためのモデル（計算式）を導き出す	正解（目的変数）はなく、大量の訓練データの構造を分析し群＝クラスタに分けるモデル（計算式）を導き出す	人工知能の予測に対する誤差をフィードバックすることで、試行錯誤を繰り返しながら精度を自動で高めていく
学習内容	説明変数（入力）と目的変数（出力）の関係	データの構造 データの類似性	報酬（誤差が少ない場合に得られるスコア）を最大化する行動
具体的な手法	回帰モデル 分類モデル	クラスタリング 主成分分析(次元削減)	Q学習 Deep Q-Network
用途の具体例	需要予測 解約予測 迷惑メールフィルタ 画像認識…	顧客特性分析 レコメンドエンジン データ分析の前処理…	株・FXなどのトレード 囲碁・将棋などのボードゲーム 自動運転や工作機械

クラスタリング

株式会社新潟人工知能研究所
技術開発部

クラスタリング

- 例：百貨店の顧客データ分析
  - 特徴量の例
    - 顧客ID、住所、氏名、性別…
    - 購入履歴、ポイント残高…
    - 購入回数、購買単価など…
  - クラスタリング（クラスタ化）
    - 特徴量をもとに、顧客をいくつかのクラスタ（グループ）に分割
    - そのクラスタにどんな意味やまとまりがあるのかは、人間が考える



分類モデルとクラスタリングの違い

分類モデル（教師あり学習）

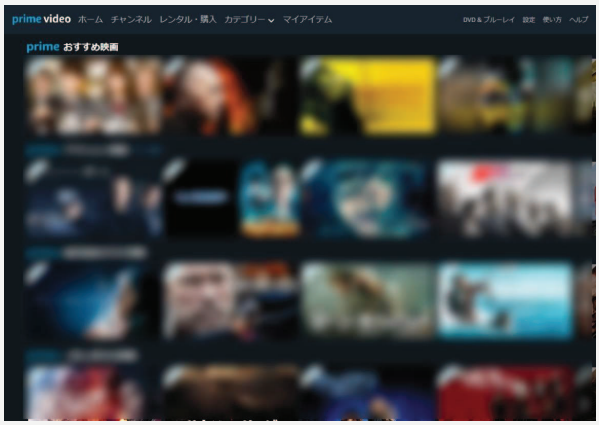
- あらかじめ決まったカテゴリのどれに属するか(の確率)を出す

クラスタリング（教師なし学習）

- どんなクラスタ＝群（まとまりの傾向）があるかを見つけ出す

クラスタリングの活用事例

- 顧客特性分析（セグメンテーション）
  - ジオグラフィック（地理的変数）
    - 国・地域・都市の規模、経済発展の度合、人口、気候、文化・生活習慣、宗教…
  - デモグラフィック（人口動態変数）
    - 年齢、性別、職業、学歴、家族構成…
  - サイコグラフィック（心理的変数）
    - 価値観、趣向、ライフスタイル…
  - ビヘイビアル（行動変数）
    - 曜日・時間、購買の状況・経路・頻度…
- キャッシュレス化で特徴量が多様化

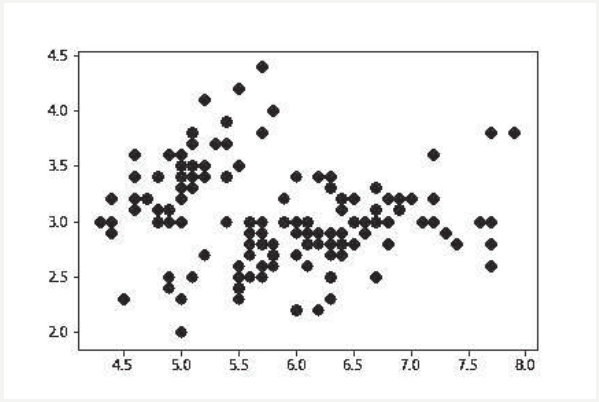


レコメンド（推薦）エンジンにも利用されるクラスタリング
Amazonの売上の35%は…　Netflixの視聴数の75%が…

参考：「Amazon.co.jp: プライム・ビデオ: Prime Video」
https://www.amazon.co.jp/Prime-Video/

クラスタリングのアルゴリズム

- k 平均法（k-means）
  - ①ランダムにクラスタの重心を選ぶ
  - ②各データから見て一番近い重心に、とりあえずクラスタリングする
  - ③各クラスタに所属するデータの重心を求め直して①に戻る（繰返し）
  - 初期値の割り振り方で結果が異なる
    - 全くのランダムよりも、狙いを付けて初期値を設定したり、初期値を何回か変えて分析を行うなどの工夫も
  - 初期化プロセスを改良 → k-means++
    - 初期の重心をできるだけ離すように



参考：YouTube「アルゴリズムがデータを分ける様子を可視化した」
https://www.youtube.com/watch?v=4F80ICKzpEU

k平均法の弱点

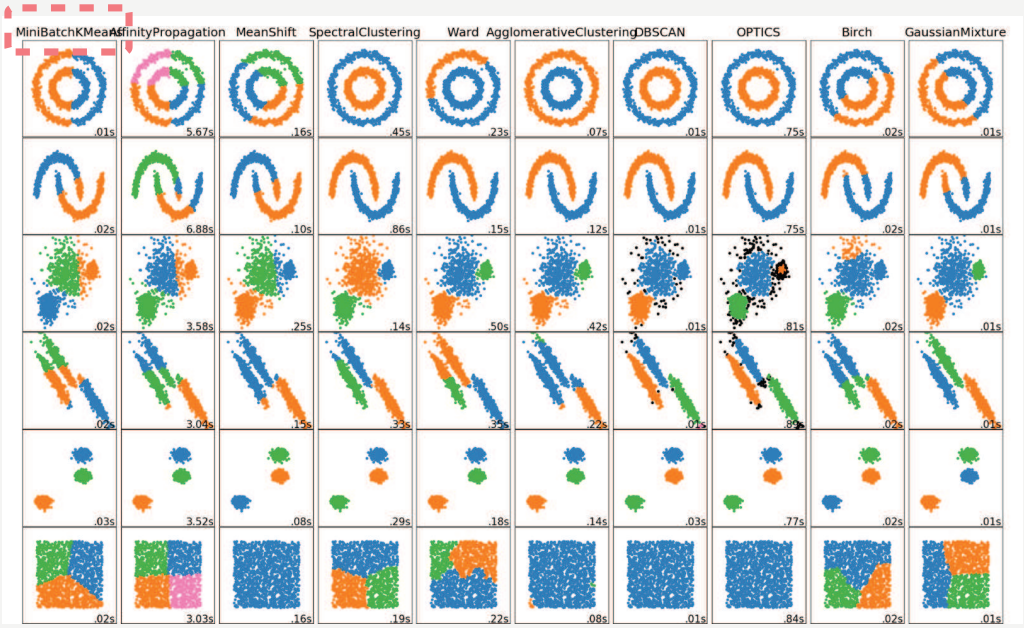
- 圧倒的人気のk平均法ですが…
  - k平均法ではクラスタは円（球）状になるため、データの分布によってはうまくいかない場合も…
  - そこで他のアルゴリズムも活用！
  - scikit-learnは多様なクラスタリングのアルゴリズムを用意しています

k平均法

VS

DBSCAN

アルゴリズムの違いによる結果例



出典：「2.3. Clustering — scikit-learn 0.21.3 documentation」
https://scikit-learn.org/stable/modules/clustering.html

次元削減

株式会社新潟人工知能研究所
技術開発部

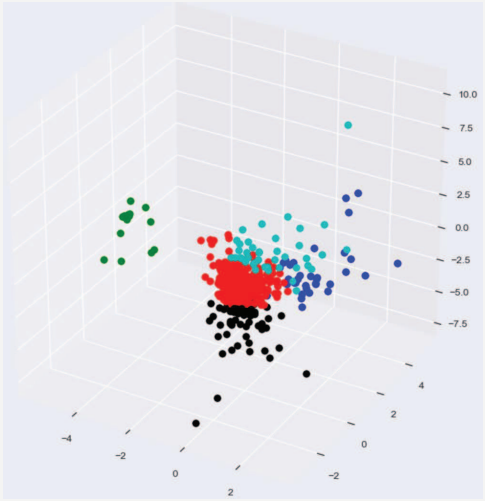
次元削減

- 次元削減（次元圧縮）
  - 特徴量（＝次元）の数を削減する
  - 利用例：
    - 学生の模試（国語・数学・社会・物理）の特徴量（4次元）を削減
    - 例えば2次元に削減すれば、理系と文系の次元になるかもしれない
    - 新しい特徴量の解釈は人間が行うため、うまく解釈できない場合も

次元削減の使いどころ

- データを可視・グラフ化したい
  - ４次元以上は視覚化が困難です…
- データの意味解釈・把握をしたい
  - 次元が多すぎると人間の限界が…
- 教師あり学習の高速化（前処理）
  - 特徴量を増やせば増やすほど、教師あり学習の精度は高くなる…かも
  - しかし組み合わせが急激に増えて学習時間が跳ね上がる（次元の呪い）
  - とはいえ「次元削減ありき」はNG

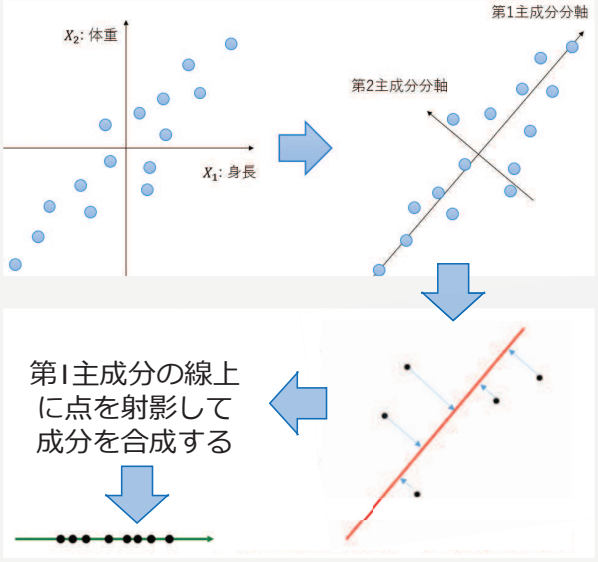
例：3次元でのクラスタリング



主成分分析（PCA：Principal Component Analysis）

- 主成分分析（PCA）
  - 相関のある多数の特徴量から全体のばらつきを最もよく表す主成分と呼ばれる特徴量を合成する手法
  - 寄与率／累積寄与率
    - 各主成分によって説明できる割合
    - 寄与率を累積すると１になる（70～80%に達する主成分数を採用）
  - 因子負荷量
    - 元の特徴量と主成分との相関係数。各主成分の意味の推定に使用する。

※詳しくはPythonの演習で確認しましょう



例）身長と体重という２次元の特徴量を、１次元の主成分（体格）に合成する

クラスタリング（Clustering）

2019/11/26.ver

クラスタリングは、特徴量をもとに、顧客をいくつかのクラスタ（グループ）に分割する教師なし学習の一種です。ここではクラスタリングの代表的な手法であるk平均法(k-means)を使用して、実習を行いましょう。

日本語の文字化け（□□□）対策

実習を始める前に「おまじない」として、グラフの文字化けを訂正するいつものプロセスを実行しておきましょう。

```
# 日本語フォントをダウンロードする。
! apt-get -y install fonts-ipafont-gothic

# キャッシュを削除する。
# 削除すべきキャッシュのファイル名は、Matplotlibがバージョンアップすると変わるため、
# 下記のrmでうまく行かない場合、! ls -ll /root/.cache/matplotlib/ でファイル名を確認
# 旧ファイル名：! rm /root/.cache/matplotlib/fontList.json
# 旧ファイル名：! rm /root/.cache/matplotlib/fontlist-v300.json

! rm /root/.cache/matplotlib/fontlist-v310.json   # 2019/10/31段階でのファイル名
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libnvidia-common-430
Use 'apt autoremove' to remove it.
The following additional packages will be installed:
  fonts-ipafont-mincho
The following NEW packages will be installed:
  fonts-ipafont-gothic fonts-ipafont-mincho
0 upgraded, 2 newly installed, 0 to remove and 7 not upgraded.
Need to get 8,251 kB of archives.
After this operation, 28.7 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/universe amd64 fonts-ipafont-gothic all 00303-18ubuntu1 [3,526 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic/universe amd64 fonts-ipafont-mincho all 00303-18ubuntu1 [4,725 kB]
Fetched 8,251 kB in 1s (6,356 kB/s)
Selecting previously unselected package fonts-ipafont-gothic.
(Reading database ... 135004 files and directories currently installed.)
Preparing to unpack .../fonts-ipafont-gothic_00303-18ubuntu1_all.deb ...
Unpacking fonts-ipafont-gothic (00303-18ubuntu1) ...
Selecting previously unselected package fonts-ipafont-mincho.
Preparing to unpack .../fonts-ipafont-mincho_00303-18ubuntu1_all.deb ...
Unpacking fonts-ipafont-mincho (00303-18ubuntu1) ...
Setting up fonts-ipafont-gothic (00303-18ubuntu1) ...
update-alternatives: using /usr/share/fonts/opentype/ipafont-gothic/ipag.ttf to provide /usr/share/fonts/truetype/fonts-
japanese-gothic.ttf (fonts-japanese-gothic.ttf) in auto mode
Setting up fonts-ipafont-mincho (00303-18ubuntu1) ...
update-alternatives: using /usr/share/fonts/opentype/ipafont-mincho/ipam.ttf to provide /usr/share/fonts/truetype/fonts-
japanese-mincho.ttf (fonts-japanese-mincho.ttf) in auto mode
Processing triggers for fontconfig (2.12.6-0ubuntu2) ...
```

ランタイムの再起動

日本語フォントを反映させるため、このタイミングでランタイムを再起動してから、

→再起動の方法：Colabメニュー → ランタイム → ランタイムを再起動

再起動後に以降のセルを実行してください

```
# 日本語フォントの設定
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(font='IPAGothic')

# memo
# plot時に↓のようにフォントを指定する方法もある
# jp_font = {'fontname':'IPAGothic'}
# plt.title('住宅価格のヒストグラム', **jp_font)
```

演習: 都道府県のクラスタリング

それでは実際に、オープンデータを用いてクラスタリングを行います。今回は各都道府県の物価データを用いて、類似している県をいくつかのグループに分類してみましょう。

ライブラリの読み込み

解析に使用するライブラリをインポートします。

```
# Pandasライブラリを、pdという別名で使用できるように宣言する
import pandas as pd
```

データセットの読み込み

政府統計の総合窓口「e-stat」から、消費者物価地域差指数というデータをダウンロードし、必要なデータを抽出したものを用意しました。次のコードセルを実行して、演習用サーバからデータをダウンロードしてください。

なお、元データはこちらからダウンロードできます。
e-stat>小売物価統計調査>構造編>10大費目別消費者物価地域差指数
政府統計の総合窓口「e-stat」

```
df_price_index= pd.read_csv('https://nai-lab.com/practice/ncc/price_index.csv')
df_price_index.head(15)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	都道府県	食料	住居	水道光熱費	家具家事用品	衣類	保険医療	交通通信	教育	教養娯楽	諸雑費
0	北海道	98.7	82.6	116.3	99.3	103.8	100.2	99.5	93.2	97.1	100.9
1	青森県	97.4	95.7	109.0	96.2	97.1	101.0	100.5	93.2	96.4	97.8
2	岩手県	96.6	89.4	111.0	102.2	97.8	100.4	99.7	90.1	99.8	97.0
3	宮城県	97.2	97.1	101.3	104.0	95.6	100.9	97.5	99.5	99.3	99.2
4	秋田県	97.3	86.1	107.2	103.1	102.4	98.9	98.0	87.1	98.0	100.2
5	山形県	101.7	91.3	111.6	93.5	105.7	97.0	99.7	105.4	99.1	97.6
6	福島県	100.0	92.5	108.6	100.9	106.2	99.9	98.8	90.9	96.6	102.9
7	茨城県	99.1	96.3	101.9	93.0	97.5	98.0	96.7	89.5	95.2	101.4
8	栃木県	99.5	87.9	96.7	100.3	115.7	99.1	97.7	101.9	94.9	99.8
9	群馬県	99.1	87.5	91.2	96.9	98.2	100.0	97.1	79.9	95.5	98.9
10	埼玉県	100.9	109.3	93.0	101.8	103.1	101.0	100.6	103.7	104.5	100.7
11	千葉県	100.6	102.0	100.4	101.0	91.7	101.1	99.1	97.9	102.0	99.6
12	東京都	103.1	133.2	94.6	102.8	96.5	101.7	104.4	108.3	104.0	99.5
13	神奈川県	102.5	124.7	97.6	101.9	101.7	99.0	104.4	112.6	104.8	102.1
14	新潟県	100.2	90.9	98.5	97.5	105.9	99.4	98.1	97.1	99.5	100.6

注釈:このデータは都道府県別に、10個の項目における物価指数が入っています。

基礎統計量

DataFrameのdescribeメソッドで、データの基礎統計量を確認します。

```
df_price_index.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	食料	住居	水道光熱費	家具家事用品	衣類	保険医療	交通通信	教育	教養娯楽	諸雑費
count	47.000000	47.000000	47.000000	47.000000	47.000000	47.000000	47.000000	47.000000	47.000000	47.00
mean	99.697872	91.408511	102.251064	99.621277	101.134043	99.861702	99.074468	96.051064	97.629787	99.87
std	2.230470	10.120724	5.615076	2.927154	5.071286	1.359436	1.771694	7.864031	3.067425	1.900
min	94.100000	80.200000	91.200000	93.000000	90.100000	97.000000	95.700000	79.900000	91.100000	94.80
25%	98.550000	85.500000	98.600000	97.800000	97.500000	99.050000	97.750000	90.500000	95.750000	98.85
50%	99.700000	88.900000	101.900000	100.000000	100.200000	100.000000	98.900000	95.200000	97.200000	99.80
75%	101.100000	95.450000	106.000000	101.500000	104.850000	100.800000	99.900000	102.050000	99.300000	101.2
max	103.500000	133.200000	116.300000	108.700000	115.700000	103.100000	104.400000	112.600000	104.800000	103.5

補足: 47都道府県のデータが、およそ100の値を中心にして存在することが確認できました。minとmaxに注目すると、特に住居は最大値と最小値の幅が大きいようです。

データの可視化

10項目のうち、「食料」と「水道光熱費」のデータを用いて、解析を進めます。まずは散布図による可視化を行って、データの分布を確認します。

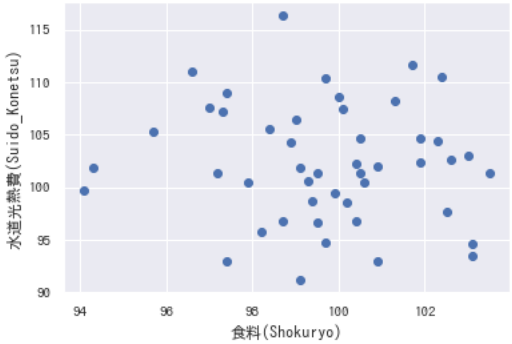
```
# 都道府県、食料、水道光熱費の列だけを抽出
df_train = df_price_index[['都道府県', '食料', '水道光熱費']]
df_train.head(15)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	都道府県	食料	水道光熱費
0	北海道	98.7	116.3
1	青森県	97.4	109.0
2	岩手県	96.6	111.0
3	宮城県	97.2	101.3
4	秋田県	97.3	107.2
5	山形県	101.7	111.6
6	福島県	100.0	108.6
7	茨城県	99.1	101.9
8	栃木県	99.5	96.7
9	群馬県	99.1	91.2
10	埼玉県	100.9	93.0
11	千葉県	100.6	100.4
12	東京都	103.1	94.6
13	神奈川県	102.5	97.6
14	新潟県	100.2	98.5

```
# プロットの準備（x軸:食料、y軸:水道光熱費）
plt.scatter(df_train['食料'], df_train['水道光熱費'])

# X軸、Y軸のラベルを設定
plt.xlabel('食料(Shokuryo)')
plt.ylabel('水道光熱費(Suido_Konetsu)')
```

```
Text(0, 0.5, '水道光熱費(Suido_Konetsu)')
```



クラスタリング

これからすべてのデータをいくつかのクラスタに分割します。まずは、先ほどのデータから都道府県の列を除外して、食料、水道光熱費だけのデータを作成します。

```
# DataFrameをDeep Copy
train_X = df_train.copy(deep=True)

# データから'都道府県'を除外
train_X = train_X.drop('都道府県',axis=1)
train_X.head(15)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	食料	水道光熱費
0	98.7	116.3
1	97.4	109.0
2	96.6	111.0
3	97.2	101.3
4	97.3	107.2
5	101.7	111.6
6	100.0	108.6
7	99.1	101.9
8	99.5	96.7
9	99.1	91.2
10	100.9	93.0
11	100.6	100.4
12	103.1	94.6
13	102.5	97.6
14	100.2	98.5

次にk平均法(k-means)で、47都道府県をクラスタリングするためのモデルを生成します。クラスタ数はとりあえず6に指定します。

```
# k平均法(k-means)のライブラリをインポート
from sklearn.cluster import KMeans

# k-meansのモデルを生成
## n_clusters = クラスタ数
## random_state = 乱数のシードを固定(デフォルト値: None)
kmodel = KMeans(n_clusters=6, random_state=0)
kmodel
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=6, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

補足: 初期値はk-means++、最大試行回数が300回、クラスタ数が4個…など、生成したモデルのパラメーターが確認できました。これらのパラメータを変更することで、結果が変化します。

参考URL: scikit-learn でクラスタ分析 (K-means 法) – Python でデータサイエンス

なお、K-Means法は一部乱数を使用することから、再現性がなかなか取りにくいのですが、random_stateを同じ値(今回の場合は0)に指定することで、必ず同じ結果が算出されます。逆に、random_stateを変えると、分類結果が変化するので試してみると面白いと思います。

```
# クラスタリングを実行
kmodel.fit(train_X)
train_Y = kmodel.labels_
print(train_Y)
```

```
[2 3 2 5 3 2 3 5 1 4 4 0 4 1 1 0 0 4 4 5 4 1 1 1 5 0 1 1 5 0 0 2 3 0 3 0 3
 3 0 3 3 2 0 3 5 5 0]
```

補足: 0から5までのクラスタ番号が47個、出力されました。本当に47個かどうかは、shapeを使用することで確認できます。

```
train_Y.shape
```

```
(47,)
```

出力されたクラスタ番号を、元のDataFrameに新しい列として連結します。

```
# 元のDataFrameをコピーして結果用のDataFrameを作成
df_results = df_train.copy(deep=True)

# クラスタ番号を新しい列として連結
df_results['クラスタ番号'] = train_Y
df_results.head(15)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	都道府県	食料	水道光熱費	クラスタ番号
0	北海道	98.7	116.3	2
1	青森県	97.4	109.0	3
2	岩手県	96.6	111.0	2
3	宮城県	97.2	101.3	5

	都道府県	食料	水道光熱費	クラスタ番号
4	秋田県	97.3	107.2	3
5	山形県	101.7	111.6	2
6	福島県	100.0	108.6	3
7	茨城県	99.1	101.9	5
8	栃木県	99.5	96.7	1
9	群馬県	99.1	91.2	4
10	埼玉県	100.9	93.0	4
11	千葉県	100.6	100.4	0
12	東京都	103.1	94.6	4
13	神奈川県	102.5	97.6	1
14	新潟県	100.2	98.5	1

さらに、DataFrameのsort_valuesメソッドで、クラスタ番号の順に並び替えてみます。

```
df_results.sort_values('クラスタ番号')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	都道府県	食料	水道光熱費	クラスタ番号
46	沖縄県	103.5	101.4	0
42	熊本県	101.9	102.3	0
38	高知県	102.6	102.6	0
35	徳島県	100.5	104.6	0
33	広島県	101.9	104.6	0
30	鳥取県	102.3	104.4	0
29	和歌山県	100.9	102.0	0
25	京都府	100.5	101.4	0
11	千葉県	100.6	100.4	0
16	石川県	103.0	103.0	0
15	富山県	100.4	102.2	0
27	兵庫県	99.9	99.5	1
26	大阪府	99.4	98.7	1
22	愛知県	98.2	95.7	1
21	静岡県	98.7	96.8	1
23	三重県	100.4	96.8	1
13	神奈川県	102.5	97.6	1
14	新潟県	100.2	98.5	1
8	栃木県	99.5	96.7	1
41	長崎県	99.7	110.4	2
2	岩手県	96.6	111.0	2
5	山形県	101.7	111.6	2
31	島根県	102.4	110.5	2

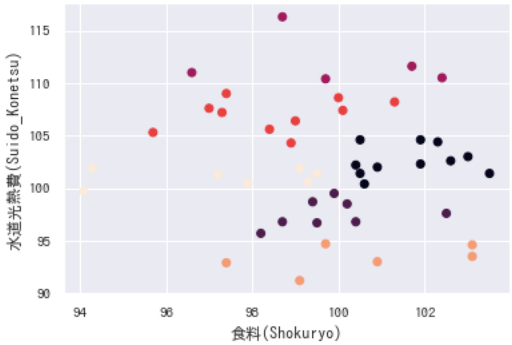
	都道府県	食料	水道光熱費	クラスタ番号
0	北海道	98.7	116.3	2
43	大分県	98.9	104.3	3
1	青森県	97.4	109.0	3
40	佐賀県	97.0	107.6	3
39	福岡県	95.7	105.3	3
37	愛媛県	100.1	107.4	3
36	香川県	98.4	105.6	3
4	秋田県	97.3	107.2	3
32	岡山県	99.0	106.4	3
34	山口県	101.3	108.2	3
6	福島県	100.0	108.6	3
20	岐阜県	97.4	92.9	4
10	埼玉県	100.9	93.0	4
18	山梨県	99.7	94.7	4
12	東京都	103.1	94.6	4
17	福井県	103.1	93.5	4
9	群馬県	99.1	91.2	4
3	宮城県	97.2	101.3	5
28	奈良県	94.3	101.9	5
7	茨城県	99.1	101.9	5
24	滋賀県	99.5	101.4	5
44	宮崎県	97.9	100.5	5
19	長野県	94.1	99.7	5
45	鹿児島県	99.3	100.6	5

補足: クラスタ番号ごとに出力することで、例えば自身が住んでいる都道府県と、食費や水道光熱費の点で近い都道府県が同じクラスたに所属されていることが確認できます。

ただ、クラスタ番号だけでは直感的な理解がしにくいのは確かです。そこで再度、散布図で可視化をします。今度はクラスタ番号ごとにプロットの色を変化させることで見分けられるようにします。matplotlibのscatterで、オプション **c=** に色番号を指定することでプロットの色を変えられます。

```
# プロットの準備（x軸:食料、y軸:水道光熱費、色:クラスタ番号 ）
plt.scatter(df_results['食料'], df_results['水道光熱費'], c=df_results['クラスタ番号'])

# X軸、Y軸のラベルを設定
plt.xlabel('食料(Shokuryo)')
plt.ylabel('水道光熱費(Suido_Konetsu)')
```

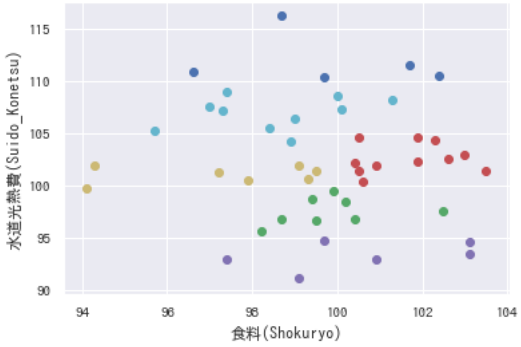
```
Text(0, 0.5, '水道光熱費(Suido_Konetsu)')
```



一部、見えにくい色がありますね。また、このままでは、どの色がどのクラスタ番号かが分かりにくいですので、カラーコードを管理するリストを作成して、もう一度プロットしてみましょう。

```
# プロット/グラフ用のカラーリストを準備
LST_COLOR = ['r','g','b','c','m','y','k','lightping','goldenrod']

# プロットの準備（x軸:食料、y軸:水道光熱費、色:カラーリストのクラスタ番号 ）
for i in range(len(df_results)):
    plt.scatter(df_results.iloc[i,1],df_results.iloc[i,2],c=LST_COLOR[df_results.iloc[i,3]],label=df_results.iloc[i,3] )

# X軸、Y軸のラベルを設定
plt.xlabel('食料(Shokuryo)')
plt.ylabel('水道光熱費(Suido_Konetsu)')
```

```
Text(0, 0.5, '水道光熱費(Suido_Konetsu)')
```



補足: カラーリストはそれぞれ、0…赤、1…緑、2…青、3…シアン、4…マゼンダ、5…黄、、、となっています。

参考: matplotlibで使用できるカラーコードは以下のサイトなどを参考にしてください。
matplotlib で指定可能な色の名前と一覧 – Python でデータサイエンス

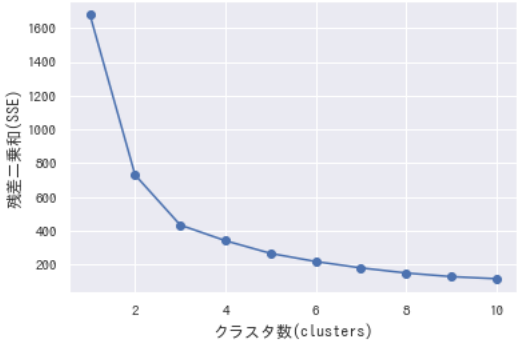
適切なクラスタ数

以上で、都道府県のクラスタリングが完了しました。 しかし、今回はクラスタ数を6としてクラスタリングしましたが、そもそも6つでよかったのでしょうか？ 実はこの6という数は適当に決めただけで何の根拠もありません。k-means法の問題点の一つは、クラスタの個数を、どう適切に指定するかというところにあり、最適なクラスタの数は試行錯誤しながら探す必要があります。

ただ、完璧な方法ではありませんが、このクラスタ数を推定するための手法として「エルボー法」や「シルエット分析」という手法があります。ここでは比較的分かりやすいな「エルボー法」についてトライしてみましょう。

k平均法のアルゴリズムを実行(fit)すると、各クラスタの重心との距離(残差二乗和)を取得することができます。そこでクラスタ数を総当り的に変化させながら、この距離をプロットすることで、適切なクラスタ数を見つけ出そうというのが、エルボー法になります。

```
# k平均法(k-means)のライブラリをインポート
from sklearn.cluster import KMeans

lst_history = []

for i  in range(1,11):                # 1～10クラスタまで一気に計算

    # k-meansのモデルを生成
    kmeans = KMeans(n_clusters=i,random_state=0)
    kmeans.fit(train_X)    # クラスタリングを実行
    lst_history.append(kmeans.inertia_)    # inertia_=残差平方和を得る

# 残差二乗和をプロット
plt.plot(range(1,11),lst_history,marker='o')
plt.xlabel('クラスタ数(clusters)')
plt.ylabel('残差二乗和(SSE)')
plt.show()
```



横軸にクラスタ数、縦軸に残差平方和を図示すると、肘（エルボー）のようにポキっと曲がった場所があることがわかります。 上記の例ではクラスタ数=3の点でグラフがポキっと曲がっていますね。 エルボー法ではこのような点を見つけることで最適な（と思われる）クラスタ数を推定します。

なお、K-meansの拡張アルゴリズムであるX-means法は、クラスタ数を自動推定する工夫がされています。 実装の参考URLを紹介しますが、クラスタ数の根拠を求められる場合は、「勘」や「なんとなく」ではなく、これらの方法を組み合わせて根拠を見つけていくことが大事です。

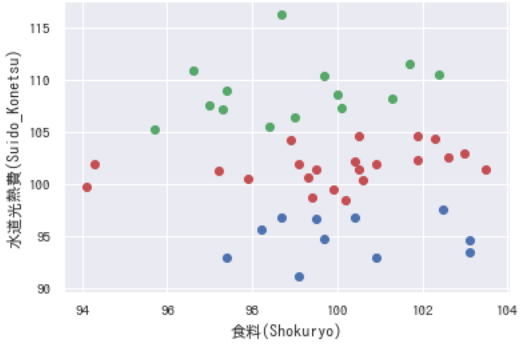
参考：x-meansでクラスタリング【python】 - Hello Wor.log
参考：クラスタ数を自動推定するX-means法を調べてみた - Qiita
参考：X-means法でクラスタ数を推定する方法 - S-Analysis

演習：適切なクラスタ数でのクラスタリング

それではせっかくですので、ここまでのまとめも兼ねて、クラスタ数を3にして、再度、クラスタリングとプロットに挑戦してみましょう。以下のプログラムの一箇所を変更して、クラスタ数3でのクラスタリングを行ってください。

```
### 演習: クラスタ数3でのクラスタリングとプロット

# Pandasライブラリを、pdという別名で使用できるように宣言する
import pandas as pd

# 都道府県、食料、水道光熱費の列だけを抽出
df_price_index= pd.read_csv('https://nai-lab.com/practice/ncc/price_index.csv')
df_train = df_price_index[['都道府県', '食料', '水道光熱費']]

# データから'都道府県'を除外
train_X = df_train.copy(deep=True)
train_X = train_X.drop('都道府県',axis=1)

# k平均法(k-means)のライブラリをインポート
from sklearn.cluster import KMeans

# k-meansのモデルを生成
## n_clusters = クラスタ数
## random_state = 乱数のシードを固定(デフォルト値: None)
kmodel = KMeans(n_clusters=3, random_state=0)

# クラスタリングを実行
kmodel.fit(train_X)
train_Y = kmodel.labels_

# 元のDataFrameをコピーして結果用のDataFrameを作成
df_results = df_train.copy(deep=True)

# クラスタ番号を新しい列として連結
df_results['クラスタ番号'] = train_Y

# プロット/グラフ用のカラーリストを準備
LST_COLOR = ['r','g','b','c','m','y','k','lightping','goldenrod']

# プロットの準備（x軸:食料、y軸:水道光熱費、色:カラーリストのクラスタ番号 ）
for i in range(len(df_results)):
    plt.scatter(df_results.iloc[i,1],df_results.iloc[i,2],c=LST_COLOR[df_results.iloc[i,3]],label=df_results.iloc[i,3] )

# X軸、Y軸のラベルを設定
plt.xlabel('食料(Shokuryo)')
plt.ylabel('水道光熱費(Suido_Konetsu)')
```

```
Text(0, 0.5, '水道光熱費(Suido_Konetsu)')
```




```
# DataFrameで結果を目視確認
df_results.sort_values('クラスタ番号')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	都道府県	食料	水道光熱費	クラスタ番号
46	沖縄県	103.5	101.4	0
27	兵庫県	99.9	99.5	0
26	大阪府	99.4	98.7	0
25	京都府	100.5	101.4	0
24	滋賀県	99.5	101.4	0
45	鹿児島県	99.3	100.6	0
30	鳥取県	102.3	104.4	0
19	長野県	94.1	99.7	0
33	広島県	101.9	104.6	0
16	石川県	103.0	103.0	0
15	富山県	100.4	102.2	0
28	奈良県	94.3	101.9	0
35	徳島県	100.5	104.6	0
14	新潟県	100.2	98.5	0
11	千葉県	100.6	100.4	0
38	高知県	102.6	102.6	0
7	茨城県	99.1	101.9	0
42	熊本県	101.9	102.3	0
3	宮城県	97.2	101.3	0
43	大分県	98.9	104.3	0
44	宮崎県	97.9	100.5	0
29	和歌山県	100.9	102.0	0
37	愛媛県	100.1	107.4	1
34	山口県	101.3	108.2	1
39	福岡県	95.7	105.3	1
32	岡山県	99.0	106.4	1
31	島根県	102.4	110.5	1

	都道府県	食料	水道光熱費	クラスタ番号
40	佐賀県	97.0	107.6	1
41	長崎県	99.7	110.4	1
36	香川県	98.4	105.6	1
0	北海道	98.7	116.3	1
6	福島県	100.0	108.6	1
5	山形県	101.7	111.6	1
4	秋田県	97.3	107.2	1
2	岩手県	96.6	111.0	1
1	青森県	97.4	109.0	1
21	静岡県	98.7	96.8	2
20	岐阜県	97.4	92.9	2
18	山梨県	99.7	94.7	2
17	福井県	103.1	93.5	2
13	神奈川県	102.5	97.6	2
12	東京都	103.1	94.6	2
10	埼玉県	100.9	93.0	2
9	群馬県	99.1	91.2	2
8	栃木県	99.5	96.7	2
22	愛知県	98.2	95.7	2
23	三重県	100.4	96.8	2

おまけ: 特徴量を10に増やしてクラスタリング

さきほどは、食料と水道光熱費という2つの特徴量による2次元の分析でしたが、特徴量を増やすことで違った結果が出力されます。ただし、人間の視覚では3次元までの認識が限界ですので、4次元以上のクラスタリングでは散布図でのプロットに併せて、他のグラフや統計量を使った分析を用います。

おまけ: さきほどの特徴量が2つ（食料、水道光熱費）のソースコードを書き換えて、10個の特徴量によるクラスタリングを実施しました。

```
# 演習: ご近所さん(都道府県)を探せ！
# 10個の特徴量を用いて3つのクラスタにクラスタリングせよ
# あなたのお住いの県と、よく似た物価の特性がある都道府県はどこだろうか？

# Pandasライブラリを、pdという別名で使用できるように宣言する
import pandas as pd

# 都道府県、食料、水道光熱費の列だけを抽出
# df_price_index= pd.read_csv('https://nai-lab.com/practice/ncc/price_index.csv')
# df_train = df_price_index[['都道府県', '食料', '水道光熱費']]

# 特徴量を削らないように書き換えた箇所（ここだけです）
df_train = pd.read_csv('https://nai-lab.com/practice/ncc/price_index.csv')

# データから'都道府県'を除外
train_X = df_train.copy(deep=True)
train_X = train_X.drop('都道府県',axis=1)

# k平均法(k-means)のライブラリをインポート
from sklearn.cluster import KMeans

# k-meansのモデルを生成
## n_clusters = クラスタ数
## random_state = 乱数のシードを固定(デフォルト値: None)
kmodel = KMeans(n_clusters=3, random_state=0)

# クラスタリングを実行
kmodel.fit(train_X)
train_Y = kmodel.labels_

# 元のDataFrameをコピーして結果用のDataFrameを作成
df_results = df_train.copy(deep=True)

# クラスタ番号を新しい列として連結
df_results['クラスタ番号'] = train_Y
```

```
# DataFrameで結果を目視確認
df_results.sort_values('クラスタ番号')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	都道府県	食料	住居	水道光熱費	家具家事用品	衣類	保険医療	交通通信	教育	教養娯楽	諸雑費	クラスタ番号
23	三重県	100.4	95.6	96.8	98.6	99.4	99.1	98.7	98.9	95.7	98.1	0
26	大阪府	99.4	97.6	98.7	99.9	99.1	99.6	101.0	108.9	102.3	96.6	0
25	京都府	100.5	92.3	101.4	100.4	98.7	97.6	102.6	112.3	101.9	101.4	0
24	滋賀県	99.5	88.3	101.4	97.6	104.3	100.6	100.6	109.0	96.8	101.1	0
22	愛知県	98.2	95.3	95.7	97.2	99.6	99.7	98.5	96.4	100.7	99.2	0
17	福井県	103.1	86.2	93.5	108.7	99.1	102.4	99.1	103.4	95.2	101.9	0
16	石川県	103.0	86.0	103.0	99.6	107.2	101.9	98.9	103.2	98.2	101.3	0
35	徳島県	100.5	89.2	104.6	101.7	107.7	98.2	97.0	97.8	98.3	99.8	0
27	兵庫県	99.9	100.0	99.5	101.7	105.4	98.4	100.8	102.2	101.5	103.5	0
11	千葉県	100.6	102.0	100.4	101.0	91.7	101.1	99.1	97.9	102.0	99.6	0
14	新潟県	100.2	90.9	98.5	97.5	105.9	99.4	98.1	97.1	99.5	100.6	0
29	和歌山県	100.9	99.0	102.0	100.0	99.5	101.3	98.9	105.1	96.8	99.6	0
43	大分県	98.9	83.8	104.3	99.6	95.6	97.2	97.1	106.5	92.9	97.8	0
8	栃木県	99.5	87.9	96.7	100.3	115.7	99.1	97.7	101.9	94.9	99.8	0
3	宮城県	97.2	97.1	101.3	104.0	95.6	100.9	97.5	99.5	99.3	99.2	0
5	山形県	101.7	91.3	111.6	93.5	105.7	97.0	99.7	105.4	99.1	97.6	0
44	宮崎県	97.9	85.2	100.5	101.3	97.5	98.9	97.8	89.6	91.1	97.2	1
30	鳥取県	102.3	80.2	104.4	100.4	105.9	100.4	97.9	88.5	94.6	99.2	1
42	熊本県	101.9	81.8	102.3	102.9	101.3	100.1	99.3	92.9	95.4	100.2	1
31	島根県	102.4	84.2	110.5	99.2	101.3	99.2	99.5	97.9	97.2	101.0	1
39	福岡県	95.7	85.3	105.3	100.2	97.0	100.0	97.2	95.8	97.2	101.7	1
38	高知県	102.6	88.9	102.6	98.2	102.0	100.8	97.5	93.6	96.8	100.5	1
34	山口県	101.3	89.1	108.2	99.6	106.4	103.1	97.4	86.0	95.5	99.5	1
41	長崎県	99.7	92.8	110.4	100.8	112.3	100.1	100.4	87.1	96.2	101.8	1
40	佐賀県	97.0	82.6	107.6	96.1	103.9	98.6	98.2	94.8	92.1	98.8	1
36	香川県	98.4	85.7	105.6	100.3	96.4	100.3	100.0	95.3	96.4	103.0	1
37	愛媛県	100.1	85.7	107.4	103.0	100.1	99.8	97.7	91.2	98.4	97.7	1
32	岡山県	99.0	86.3	106.4	100.2	106.6	101.2	95.7	87.4	96.0	101.2	1
33	広島県	101.9	89.1	104.6	97.2	95.5	99.9	99.3	97.6	96.4	99.4	1
0	北海道	98.7	82.6	116.3	99.3	103.8	100.2	99.5	93.2	97.1	100.9	1
45	鹿児島県	99.3	80.5	100.6	94.1	90.1	100.4	99.2	91.3	92.3	96.9	1
1	青森県	97.4	95.7	109.0	96.2	97.1	101.0	100.5	93.2	96.4	97.8	1
2	岩手県	96.6	89.4	111.0	102.2	97.8	100.4	99.7	90.1	99.8	97.0	1
4	秋田県	97.3	86.1	107.2	103.1	102.4	98.9	98.0	87.1	98.0	100.2	1
6	福島県	100.0	92.5	108.6	100.9	106.2	99.9	98.8	90.9	96.6	102.9	1
7	茨城県	99.1	96.3	101.9	93.0	97.5	98.0	96.7	89.5	95.2	101.4	1

	都道府県	食料	住居	水道光熱費	家具家事用品	衣類	保険医療	交通通信	教育	教養娯楽	諸雑費	クラスタ番号
9	群馬県	99.1	87.5	91.2	96.9	98.2	100.0	97.1	79.9	95.5	98.9	1
28	奈良県	94.3	84.2	101.9	98.9	93.3	99.3	100.4	95.2	99.3	99.3	1
46	沖縄県	103.5	84.8	101.4	99.4	100.2	100.8	98.2	93.6	95.8	94.8	1
15	富山県	100.4	87.3	102.2	101.7	100.0	101.8	97.7	85.7	97.2	102.6	1
18	山梨県	99.7	94.3	94.7	100.2	107.2	97.4	98.4	87.7	98.3	100.0	1
19	長野県	94.1	87.1	99.7	97.0	102.5	98.3	99.8	94.7	98.1	102.6	1
20	岐阜県	97.4	84.1	92.9	94.1	101.9	99.3	100.4	92.5	97.4	99.8	1
21	静岡県	98.7	97.2	96.8	98.0	97.4	100.2	99.5	82.0	99.9	98.4	1
13	神奈川県	102.5	124.7	97.6	101.9	101.7	99.0	104.4	112.6	104.8	102.1	2
12	東京都	103.1	133.2	94.6	102.8	96.5	101.7	104.4	108.3	104.0	99.5	2
10	埼玉県	100.9	109.3	93.0	101.8	103.1	101.0	100.6	103.7	104.5	100.7	2

この結果は、先ほどと大きく異なるかと思います。今回は10個の特徴量において計算を行なっているため、散布図でプロットしようとしても、10次元を可視化することは不可能です。

それでは、この分類が正しいかどうかは何が証明してくれるのでしょうか。実は、そこがクラスタリングなどの教師なし学習の難しい部分で、正しいかどうかはあくまでも人間の主観的な部分に依存してしまいます。そのため、客観的かつ決定的な証拠を突きつけることは難しいです。ただ一方で、今回のように正解データがなくても簡単に扱えるため、探索的にデータの規則性を見つけるには有効な手段と言えるでしょう。

次元削減／主成分分析(PCA)

2019/11/26.ver
特徴量（次元）を減らす次元圧縮を行います。主成分分析(PCA)を実施しながら、何次元まで削減するのかが妥当であるのかを探索しながら、進めていきましょう。

日本語の文字化け（□□□）を訂正

実習を始める前に「おまじない」として、グラフの文字化けを訂正するいつものプロセスを実行しておきましょう。

```
# 日本語フォントをダウンロードする。
! apt-get -y install fonts-ipafont-gothic

# キャッシュを削除する。
# 削除すべきキャッシュのファイル名は、Matplotlibがバージョンアップすると変わるため、
# 下記のrmでうまく行かない場合、! ls -ll /root/.cache/matplotlib/ でファイル名を確認
# 旧ファイル名: ! rm /root/.cache/matplotlib/fontList.json
# 旧ファイル名: ! rm /root/.cache/matplotlib/fontlist-v300.json

! rm /root/.cache/matplotlib/fontlist-v310.json   # 2019/10/31段階でのファイル名
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libnvidia-common-430
Use 'apt autoremove' to remove it.
The following additional packages will be installed:
  fonts-ipafont-mincho
The following NEW packages will be installed:
  fonts-ipafont-gothic fonts-ipafont-mincho
0 upgraded, 2 newly installed, 0 to remove and 7 not upgraded.
Need to get 8,251 kB of archives.
After this operation, 28.7 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/universe amd64 fonts-ipafont-gothic
all 00303-18ubuntu1 [3,526 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic/universe amd64 fonts-ipafont-mincho
all 00303-18ubuntu1 [4,725 kB]
Fetched 8,251 kB in 2s (4,837 kB/s)
Selecting previously unselected package fonts-ipafont-gothic.
(Reading database ... 135004 files and directories currently installed.)
Preparing to unpack .../fonts-ipafont-gothic_00303-18ubuntu1_all.deb ...
Unpacking fonts-ipafont-gothic (00303-18ubuntu1) ...
Selecting previously unselected package fonts-ipafont-mincho.
Preparing to unpack .../fonts-ipafont-mincho_00303-18ubuntu1_all.deb ...
```

```
Unpacking fonts-ipafont-mincho (00303-18ubuntu1) ...
Setting up fonts-ipafont-gothic (00303-18ubuntu1) ...
update-alternatives: using /usr/share/fonts/opentype/ipafont-gothic/ipag.ttf to
provide /usr/share/fonts/truetype/fonts-japanese-gothic.ttf (fonts-japanese-
gothic.ttf) in auto mode
Setting up fonts-ipafont-mincho (00303-18ubuntu1) ...
update-alternatives: using /usr/share/fonts/opentype/ipafont-mincho/ipam.ttf to
provide /usr/share/fonts/truetype/fonts-japanese-mincho.ttf (fonts-japanese-
mincho.ttf) in auto mode
Processing triggers for fontconfig (2.12.6-0ubuntu2) ...
```

ランタイムの再起動

日本語フォントを反映させるため、このタイミングでランタイムを再起動してから、

→再起動の方法：Colabメニュー → ランタイム → ランタイムを再起動

再起動後に以降のセルを実行してください

```
# 日本語フォントの設定
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(font='IPAGothic')

# memo
# plot時に↓のようにフォントを指定する方法もある
# jp_font = {'fontname':'IPAGothic'}
# plt.title('住宅価格のヒストグラム', **jp_font)
```

成績データの次元削減

今回は学生の定期テスト(5教科)の結果について、次元削減を行います。国語、数学、英語、理科、社会といった各教科の得点から、例えば「理系」「文系」のように、人間にも分かりやすい特徴にまとめることができるかを挑戦します。

データの準備

DataFrameにA～Hまでの8名の5教科分の成績データを作成します。

```
import numpy  as np   # Numpy ライブラリをnpという別名で宣言
import pandas as pd   # Pandasライブラリをpdという別名で宣言

# DataFrameに成績データを作成
df_data = pd.DataFrame([
    ['A', 60, 85, 40, 90, 50],
    ['B', 80, 55, 85, 70, 90],
    ['C', 35, 85, 40, 90, 50],
    ['D', 50, 40, 45, 60, 55],
    ['E', 45, 90, 55, 85, 60],
```

```
    ['F', 80, 55, 75, 65, 85],
    ['G', 82, 90, 85, 85, 90],
],columns=["氏名", "国語", "数学", "英語", "理科", "社会"])

df_data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	氏名	国語	数学	英語	理科	社会
0	A	60	85	40	90	50
1	B	80	55	85	70	90
2	C	35	85	40	90	50
3	D	50	40	45	60	55
4	E	45	90	55	85	60
5	F	80	55	75	65	85
6	G	82	90	85	85	90

5教科の特徴量で主成分分析を行う際に、氏名の列は必要ありませんので、DataFrameのdropメソッドで列を除外しておきます。

```
# 分析用にDataFrameをコピー
train_X = df_data.copy(deep=True)

# データから'氏名'の列を除外
train_X = train_X.drop('氏名',axis=1)
train_X
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	国語	数学	英語	理科	社会
0	60	85	40	90	50
1	80	55	85	70	90
2	35	85	40	90	50
3	50	40	45	60	55
4	45	90	55	85	60
5	80	55	75	65	85
6	82	90	85	85	90

主成分分析器の生成

scikit-learnのPCAをインポートして、主成分分析器を生成します。分析器のオプションとして、削減後の次元数を指定する必要があります。そもそもまだ何次元に削減すべきか定まってませんが、まずは試しに2次元に圧縮してみます。

```
# 主成分分析(PCA)のライブラリをインポート
from sklearn.decomposition import PCA

# 主成分分析器の生成(次元数=2)
pca = PCA(n_components = 2)
pca.fit(train_X) # 主成分分析を実行
```

```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

寄与率の確認

pcaのexplained_variance_ratio_で、主成分ごとの寄与率を確認します。寄与率とは、それぞれの主成分によって、元の特徴量（ここでは5教科）を、どれだけの割合で表現できているかを表します。

```
# 寄与率の確認
pca.explained_variance_ratio_
```

```
array([0.6627864, 0.3009016])
```

注釈: 第1主成分だけで、元の5教科の特徴量の約66%が表現できることが分かりました。また、第2主成分には30%の寄与率があることが確認できます。

```
# 累計寄与率
np.cumsum(pca.explained_variance_ratio_)
```

```
array([0.6627864, 0.963688 ])
```

注釈: pcaのexplained_variance_ratio_では、寄与率を累積した値を確認できます。2番目の数値が約96%となっており、先程の寄与率66%と30%の合計値になっていることが確認できます。

適切な次元の決定

経験則的に、この累積寄与率が70～80%に達するところの主成分数を採用することが多いようです。今回は2つの次元で、もとの5教科の特徴量を96%まで表現できることが分かりましたので、このまま2次元への削減を続けたいと思います。なお、主成分分析器を生成する際に、オプションとしてn_componentsに0～1までの実数を指定すると、累積寄与率がその値になるまで主成分を求めることができます。今回のケースですと、累積寄与率80%で良しとするのであれば、以下のように書き直すことも可能です。

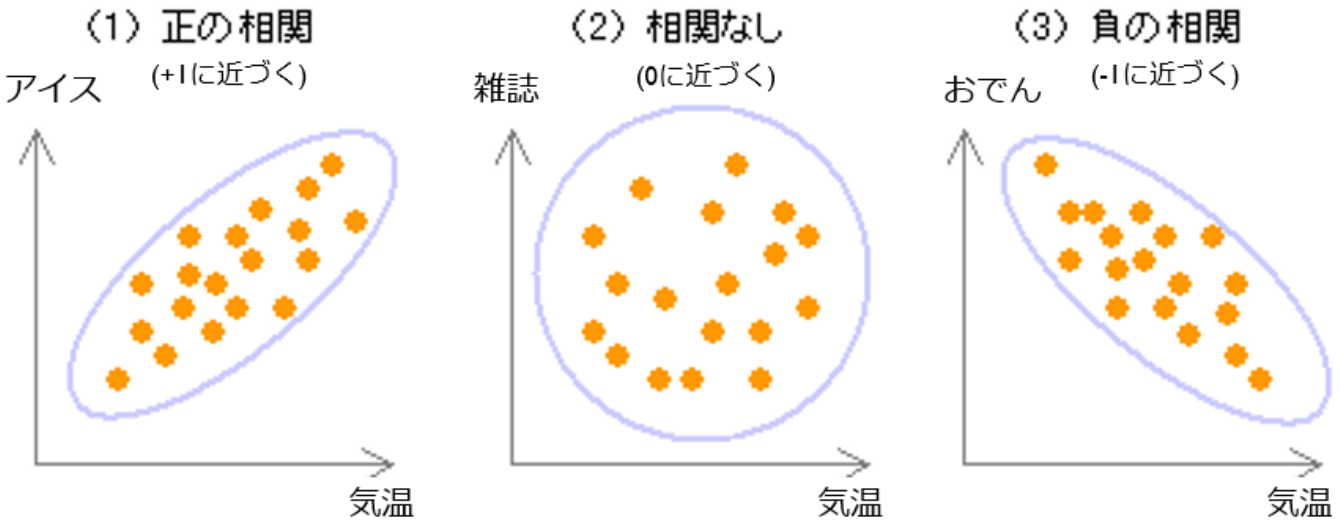
```
# 主成分分析器の生成(累積寄与率80%超)
pca = PCA(n_components = 0.8)
pca.fit(train_X) # 主成分分析を実行

np.cumsum(pca.explained_variance_ratio_) # 累計寄与率
```

```
array([0.6627864, 0.963688 ])
```

注釈: 2つの主成分で80%を超えますので、先程と同じ結果になりました。

因子負荷量

各特徴量と各主成分との相関係数を表す値に因子負荷量があります。これは各主成分にどんな意味が含まれているのかを、人間側が推定する手がかりになる数値です。 相関係数ですので、-1から1までの値を取り、絶対値が1に近いほど、強い関連があることを表しています。以下に、正の相関・相関なし・負の相関の例を図示しましたので参考にしてください。



pcaのcomponents_で、因子負荷量を確認できます。

```
# 因子負荷量の確認
pca.components_
```

```
array([[-0.5292785 ,  0.24093821, -0.5759486 ,  0.19419145, -0.54072724],
       [-0.12716349, -0.83408142, -0.22878079, -0.45588835, -0.16722078]])
```

少々、見にくいですので、元のDataFrameからcolumns(列名)を連結して見やすくします。

```
# 因子負荷量の確認
pd.DataFrame(pca.components_ , index=["第1主成分", "第2主成分"]
,columns=list(train_X.columns))
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	国語	数学	英語	理科	社会
第1主成分	-0.529278	0.240938	-0.575949	0.194191	-0.540727
第2主成分	-0.127163	-0.834081	-0.228781	-0.455888	-0.167221

注釈: 第1主成分の絶対値を確認すると、国語、英語、社会の相関が強め(-0.52と-0.57と-0.54)ため、これはいわゆる「文系」科目の特徴を表していると思われます。 また、第2主成分の絶対値では、数学の相関が圧倒的に強く(-0.83)、次いで理科との相関が強い(-0.45)です。これは「理系」科目の特徴を表現しているのでは？と推測できます。

次元削減の実行

pcaのtransformメソッドで、次元削減を実行します。結果はnumpyのndarray型で返却されますが、少し見づらいので、ここではDataFrameで行名、列名を付加して見栄えを整えています。

```
# 次元削減の実行
train_Y = pca.transform(train_X)

# DataFrameで見栄えを調整
pd.DataFrame(train_Y,index=list(df_data['氏名']) ,columns=["第1主成分", "第2主成分"])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	第1主成分	第2主成分
A	28.507690	-8.792910
B	-40.736631	5.820064
C	41.739653	-5.613822
D	11.549133	41.709032
E	22.634100	-13.880342
F	-33.244466	11.223417
G	-30.449479	-30.465438

注釈: A～Gの8名について、それぞれ第1主成分、第2主成分のスコアが取得できました。これで、国語、数学、英語、理科、社会の5次元の特徴量を、2次元の特徴量まで削減できました。

では本当に、主成分が元の特徴量を表現しているかを検証するために、元データに主成分の列を連結してみましょう。

```
# 元のDataFrameをコピーして結果用のDataFrameを作成
df_result = df_data.copy(deep=True)

df_result['文系スコア？'] = pd.Series(train_Y[ : , 0])
df_result['理系スコア？'] = pd.Series(train_Y[ : , 1])

df_result
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	氏名	国語	数学	英語	理科	社会	文系スコア？	理系スコア？
0	A	60	85	40	90	50	28.507690	-8.792910
1	B	80	55	85	70	90	-40.736631	5.820064
2	C	35	85	40	90	50	41.739653	-5.613822
3	D	50	40	45	60	55	11.549133	41.709032
4	E	45	90	55	85	60	22.634100	-13.880342
5	F	80	55	75	65	85	-33.244466	11.223417
6	G	82	90	85	85	90	-30.449479	-30.465438

注釈: 元の5つの特徴量と、新たに得た2つの特徴量を並べてみました。

しかし、違和感があるかもしれません。例えば、Bさんは国語・英語・社会で高得点を取っているのにも関わらず、文系スコアだと思われる第1主成分が-40.73...になっています。また、逆にDさんは、数学も理科も得点が低いのに、理系のスコアだと推測した第2主成分が、41.70...ととても高いです。

実は、これにはカラクリがあって、因子負荷量を出力したときに、それぞれ相関係数が負（マイナス）になっていました。つまり、この文理のスコアは、マイナスであればあるほど優秀ということになります。この正・負を読み間違えると、まったく真逆の意味になりますので、注意が必要です。

新しい特徴量でプロット

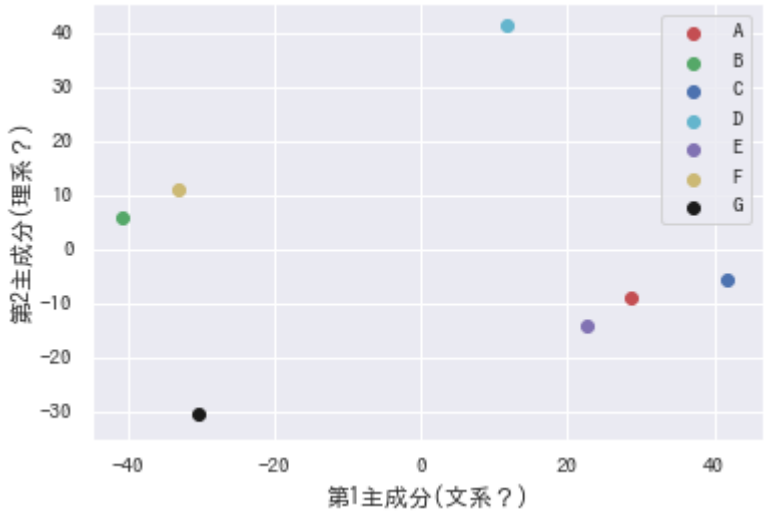
視覚化しやすい2次元になりましたので、せっかくですからプロットしてみます。

参考: プロットの色分けをするのに、プログラム中で、カラーコードのリストを用意しました。matplotlibで使用できるカラーコードは以下のサイトなどを参考にしてください。
matplotlib で指定可能な色の名前と一覧 – Python でデータサイエンス

```
# Matplotlibライブラリを、pltという別名で使用できるように宣言する
import matplotlib.pyplot as plt

# プロット/グラフ用の色番号リストを準備
LST_COLOR = ['r','g','b','c','m','y','k','lightping','goldenrod']

# 凡例用に氏名の列を取得
name = df_data['氏名']

# 色を変えながらデータを1件ずつプロット
for i in range(len(name)):
    plt.scatter(train_Y[i,0],train_Y[i,1] , c=LST_COLOR[i] , label=name[i])

# X軸、Y軸のラベルを付ける
plt.xlabel('第1主成分(文系？)')
plt.ylabel('第2主成分(理系？)')

plt.legend()   # 凡例を付ける
df_result
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	氏名	国語	数学	英語	理科	社会	文系スコア？	理系スコア？
0	A	60	85	40	90	50	28.507690	-8.792910
1	B	80	55	85	70	90	-40.736631	5.820064
2	C	35	85	40	90	50	41.739653	-5.613822
3	D	50	40	45	60	55	11.549133	41.709032
4	E	45	90	55	85	60	22.634100	-13.880342
5	F	80	55	75	65	85	-33.244466	11.223417
6	G	82	90	85	85	90	-30.449479	-30.465438



もとの5次元の特徴量と、削減後のプロットを比較しながら、上手に次元削減ができているかを確認しましょう。

まとめ

以上で主成分分析(PCA)を用いた次元削減を行いました。

この演習の発展としては、たとえば都道府県別の10の特徴量を持った消費者物価地域差指数を、3次元や2次元に次元削減してからクラスタリングして、プロットしてみるというのも面白いと思います。

ただし、繰り返しになりますが、いきなり「次元削減ありき」で考えずに、ケース・バイ・ケースで適切に使用するように気をつけてください。

AI2-4

AIの実装（自然言語処理）

－ 講義内容 －

- 活用事例
- 技術解説
- 演習：形態素解析と文章のネガポジ判定
- 演習：単語のベクトル化

自然言語処理（Natural Language Processing）

2019年11月27日（水）

新潟人工知能研究所 技術開発部

大目次

p03. 科目概要
p05. 活用事例
p15. 技術解説

科目概要

株式会社新潟人工知能研究所
技術開発部

シラバス

- 自然言語処理（NLP:Natural Language Processing）の概要を理解する。
- NLPに関する一連の流れ（形態素解析、単語意味解析、構文解析、文章意味解析、文脈解析）と、それを実装するライブラリ群について理解する。
- また、MeCabやgensimライブラリを用いて形態素解析や単語意味解析を、Pythonで実装する。

活用事例

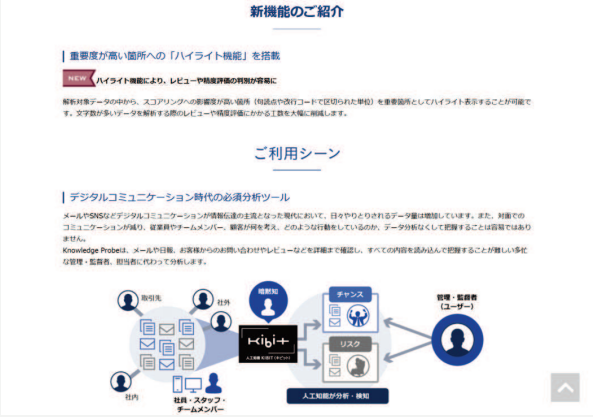
株式会社新潟人工知能研究所
技術開発部

多彩な活用シーン

- 処理対象が幅広く汎用性も高い
  - メール、チャット、連絡文
  - 論文、契約書、同意書
  - SNS上のコメント、レビュー
  - 音声を変換したテキスト
  - 業務報告書、面接記録
  - 電子カルテ、申し送り、などなど

営業支援・日報分析

- 受注チャンスと失注リスク抽出
  - 販売部門とお客様との日々の連絡、メール、日報などの報告書を分析。
  - 毎日のメールや日報の数が多すぎて詳細までチェックが困難な管理者に代わり、受注機会に発展しうる情報や失注リスクを見つける。



引用： FRONTEO「Knowledge Probe - 人工知能がビジネスチャンスやリスクを検知」
http://www.kibit-platform.com/products/knowledge-probe/

人事（HR Tech：Human Resources）

- 人事領域の課題解決
  - 人工知能が従業員のメールを分析
  - 内容から会社への不平不満やストレス、パワハラの疑い等を検知
  - 予兆を早期に発見し、人材流出の防止とハラスメントの防止、組織診断、管理職・後継者育成を支援
- 新卒採用エントリーシート選考の適正・効率化
  - http://www.fronteo.com/wp-content/themes/FRONTEO//corporate/news/uploadfile/docs/20180405.pdf
- 早期退職の防止
  - http://www.fronteo.com/wp-content/themes/FRONTEO//corporate/news/uploadfile/docs/20170309.pdf

引用： FRONTEO「人事領域の課題解決」
http://www.kibit-platform.com/solution/hr-management/

法務／契約

- 膨大なテキストを扱う法務は、自然言語処理による効率化が期待される分野
- 米国207社 法務部門に対する意識調査
  - 「企業法務部門でAI活用が主流になるのはいつ頃か」との質問に「5年以内」は21%、「10年以内」は、39%にのぼった。（大手調査会社トムソン・ロイター）
- 法務契約書の内容を自動でチェック
  - 記載されるべき条項の欠如や不適合など

参考：LegalForce - AIによる契約書レビュー支援サービス
https://legalforce-cloud.com/

医療／福祉

- カルテの整理・管理・診断支援
  - 医療機関が抱える大量の医療データを病名やキーワードで整理する
  - 疾病と症状のあいだにある関係を明らかにする
  - 明らかになった関係を参考にして、より高品質な診断を可能にする

clinical_text
今朝から突然の腰痛があり、吐き始めた。お腹の痛みなし。既往歴に、尿路結石、1型糖尿病。
昨日から腹痛があり、本日発熱あり。嘔気あったが、吐かなかった。
3日前から体のだるさがあったが、熱はなかった。本日は息苦しさがあり、一度吐いたが、気分はよくならなかった。既往歴に喘息、糖尿病あり。
4週間前から咳、痰が続いている。熱(-), 鼻汁(-) 既往歴: 緩徐進行型1型糖尿病

positive_symptom	negative_symptom	past_history
腰痛,嘔吐	腹痛	尿路結石,1型糖尿病
腹痛,発熱,嘔気	嘔吐	
倦怠感,呼吸苦,嘔吐	発熱	気管支喘息,糖尿病
咳嗽,喀痰	発熱,鼻汁	緩徐進行型1型糖尿病

参考：医師のカルテから、陽性／陰性症状・既往歴を抽出する自然言語処理の例

小売／顧客対応

- マナミさん（アスクル）
  - お客様サポート用チャットボット
  - 『マナミさんは、人間に換算すると9人月の働きをしている』（アスクル 横田氏）
  - 『IBM Watsonをベースとした対話システム「バーチャルエージェント」を、サイト内のマナミさんの会話エンジンに導入』（ソフトバンク プレスリリース）

引用：ネットショップ担当者フォーラム
「9人月分の働きをするアスクルのAIチャットボット
「マナミさん」「アオイくん」運用の裏側」
https://netshop.impress.co.jp/node/6503

チャットボット（雑談）

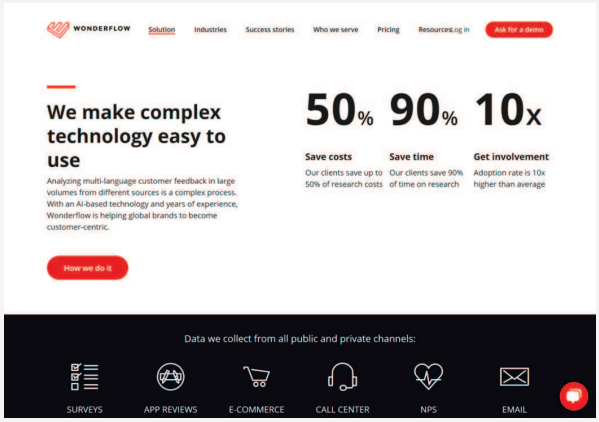
- りんな（マイクロソフト）
  - 『人間と同じように、文脈を踏まえた適切な対応で、自然な会話を続けることが可能に』
  - 『ソーシャルAIチャットボットりんなにおいて、本日から最新の会話エンジン「共感モデル（Empathy model）」(アルファ版)を採用』

↑左はローソンのAI「あきこ」さんで、右はマイクロソフトの女子高生AI「りんな」さん。どちらもLINEで楽しいおしゃべりができます

引用：マイクロソフト「ソーシャル AI チャットボット「りんな」に最新会話エンジン「共感モデル」を採用 - News Center Japan」
https://news.microsoft.com/ja-jp/2018/05/22/180522-rinna-empathy-model/

センチメント（感情）分析

- 主な機能
  - 自社の製品やサービスに関するレビューをSNS等から収集
  - 収集したレビューに対してセンチメント分析（感情分析）を実行
  - 否定的なレビューを投稿したユーザの属性や、好意的なレビューにおけるキーワードを取得・分析



参考：All aspects of customer feedback analysis delivered – Wonderflow
https://www.softbank.jp/corp/group/sbm/news/press/2017/20170428_01/

パーソナリティ（性格）診断

- Personality Insights（IBM）
  - 公式サイトより「テキストから筆者の性格を推定してみましょう。Personality Insightsは、言語学的分析とパーソナリティ理論を応用し、テキストデータから、その筆者の特徴を推測します」
  - 任意のテキストやTwitterアカウントから、その人の性格特性を算出
  - IBM Cloud（旧Bluemix）にて提供

参考：IBM Watoson「Personality Insights」
https://personality-insights-demo.ng.bluemix.net/
https://www.youtube.com/watch?v=VBhA8q9J52M

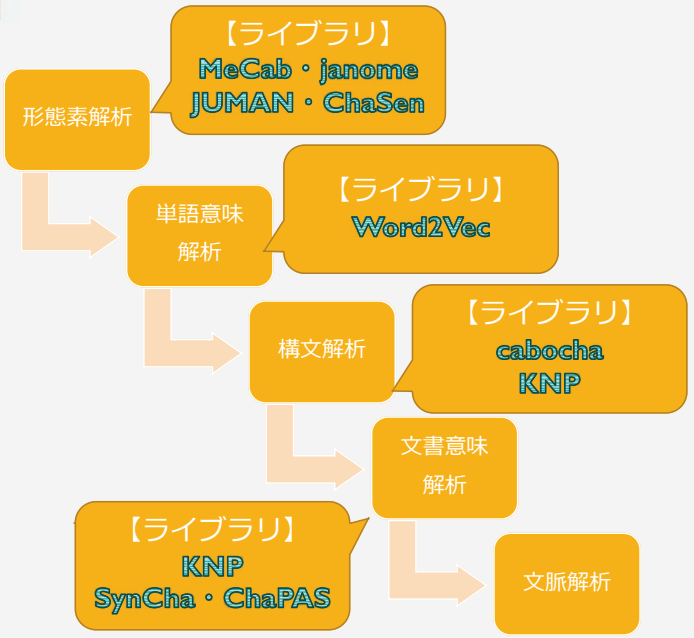
技術解説

株式会社新潟人工知能研究所
技術開発部

自然言語処理/NLP

人間が日常的に使っている自然な言語体系をコンピュータに処理させる一連の技術

- ステップ１【単語単位】形態素解析
  - 単語分割＆品詞タグ付け
  - 僕／は／美しい／花／が／好き／です
- ステップ２【単語単位】単語意味解析
  - 曖昧性の解消 & 単語の意味付け
  - 美しい≒きれい／花≒お花≒花畑？
- ステップ３【文章単位】構文解析
  - 係り受け＝美しい(修飾語)→花(主語)
- ステップ４【文章単位】文章意味解析
- ステップ５【文書単位】文脈解析



参考．自然言語処理の流れ

形態素解析

- 単語の分割と品詞のタグ付け
  - 検索エンジンでも使われている
- 実装用のライブラリ（日本語）
  - MeCab
  - Janome、JUMAN、ChaSen、など
- 新語や固有表現を含んだ辞書
  - 恋ダンス ≠ 「恋」「ダンス」
  - mecab-ipadic-Neologd 辞書ファイル

- 形態素解析の例
  - 僕は美しい花が好きです
    - 僕…名詞
    - は…助詞
    - 美しい…形容詞
    - 花…名詞
    - が…助詞
    - 好き…形容詞
    - です…助動詞

単語意味解析と訓練データ

- 人工知能に単語意味を考えさせるには適切な文章を訓練データとしてモデルを作る必要がある
- 目的に合った訓練データを準備
  - 専門的すぎる用語、俗語、など
    - ウィキペディアには載ってない…
  - １９８０年代と現代の「ヤバい」
    - SNS上のトレンド分析しようと…
- 「日本語」を扱うことの留意点

名称	概要	データ量
Wikipedia日本語全文データ	世界最大のフリー百科事典。日本語全文データを公開	3.7GB
京都大学テキストコーパス	毎日新聞の記事や社説から約4万文に対して人手で形態素・構文情報を付与	4万語
京都大学ウェブ文書リードコーパス	ウェブのニュース記事、百科事典記事、ブログなどのリード文に対して、人手で言語情報を付与	5000語
青空文庫 形態素解析データ集	青空文庫に収録されている著作に文に対して形態素解析を実行した結果をまとめ	約11000語
Twitter日本語評判分析データセット	携帯電話等に関してつぶやいている2015年から2016年ごろのツイートに対して、人手で感情ラベルを付与	約53万語

参考：日本語で提供されているコーパス一覧

機械学習で自然言語を扱う手法①

- 発想：文章中に出てくる単語の出現頻度によって、その文章の言わんとしていることの特徴量とするのことができるのでは？
- Bow（Bag-of-words）
  - ①形態素解析して単語に分ける
  - ②ストップワードを取り除く
    - 処理対象から除外する言葉のこと
  - ③単語にIDを付番してリスト化
  - ④文中の単語の出現頻度を数える

A：私はラーメンが好きです。
B：私は餃子が好きです。
C：私はラーメンが嫌いです。

単語	ID	出現頻度
私	1	3
は	-	-
ラーメン	2	2
が	-	-
好き	3	2
です	-	-
餃子	4	1
嫌い	5	1

機械学習で自然言語を扱う手法②

- 発想：単語をベクトル化することで、単語の持つ意味を数式でモデル化が可能になるのでは？
  - Gensimライブラリ word2vecモデル
  - 膨大な自然言語データ（コーパス）を解析し、対象単語の周辺にある単語は、関係性が強いとする
  - 単語間の類似性や計算が可能に！
    - ”王様”－”男”＋”女”＝「女王」
    - “イチロー”－”野球”＋”サッカー”

	イチロー	プログラマー	間食	新潟	…
勤勉	0.99	0.85	0.11	0.82	
怠惰	0.01	0.88	0.67	0.21	
運動	0.94	0.05	0.35	0.77	
場所	0.06	0.03	0.02	0.95	
食事	0.22	0.33	0.55	0.78	
…					
…					

参考：word2vecによるベクトル化のイメージ

自然言語処理（NLP:natural language processing）

自然言語処理は、人間が日常的に使っている自然な言語体系をコンピュータに処理させる一連の技術です。

形態素解析→単語意味解析→構文解析→文章意味解析→文脈解析などのステップを踏み、様々な技術が組み合わさった総合格闘技のような分野です。

回帰、分類などに比べて、遥かに難易度が高く画像認識などにおいて高精度を実現しているディープラーニングなどにとっても、自然言語処理は「最後のフロンティア」と表現する専門家がいるほど、まだまだ発展の余地はあります。

形態素解析（MeCab）

日本語の自然言語処理をする場合、まず行わなければいけない処理が「形態素解析」です。

例えば、「私は台所で料理します」という文章を形態素解析すると 「私(代名詞)/は（副助詞）/台所(名詞)/で(助詞)/料理（名詞）/し(動詞)/ます（助動詞）」というように言葉を分割していきます。

形態素とは、言葉が意味を持つまとまりの単語の最小単位のことです。英語をはじめとする世界中の言語は、単語と単語の間に空白を入れる「分かち書き」をするものがほとんどですが、日本語は分かち書きを行わず、またかな漢字混じりの文章のため、英語などと比べていきなり、自然言語処理のハードルが高いのです。

しかし、この形態素解析はキホンの技術であるがゆえに、日本語の自然言語処理では様々なシーンで活躍しますので、ぜひマスターしましょう。

MeCabのインストール

日本語の形態素解析にはいくつかのツールがありますが、定番として多くの人が「MeCab（めかぶ）」を利用しています。 MeCabは解析精度が高く、実行速度も速いため、さまざまな場面で利用されており、また後述のMeCab用の外部辞書を利用することで、さらに高精度が期待できます。

ColaboratoryでMeCabを利用するには、aptコマンドでインストールを行う必要がります。以下のコードを実行して形態素解析の環境を構築しましょう。なお、インストール完了まで2～3分程度かかります。

```
# 形態素解析(MeCab)に必要なライブラリをインストールする
! apt install aptitude swig
! aptitude install mecab libmecab-dev mecab-ipadic-utf8 git make curl xz-utils
file -y
! pip install mecab-python3
```

```
～　出力省略　～

done!
Setting up mecab-jumandic (7.0-20130310-4) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

```
Processing triggers for libc-bin (2.27-3ubuntu1) ...

Collecting mecab-python3
□[?25l  Downloading
https://files.pythonhosted.org/packages/bf/c5/acaa5eeb3f61a98cf31d8aacbbbf23f3e1c9
eadfe6e2d508add82e7e9039/mecab_python3-0.996.3-cp36-cp36m-manylinux2010_x86_64.whl
(17.1MB)
□[K     |████████████████████████████████| 17.1MB 245kB/s
□[?25hInstalling collected packages: mecab-python3
Successfully installed mecab-python3-0.996.3
```

補足: インストールの進捗バーが止まるまで、しばらく(2〜3分程度）お待ちください。

インポートとMeCabの生成

インストールが完了すれば、あとはMeCabをインポートするだけで、すぐに形態素解析が使用できます。

まずは、何もオプションを付けずに、MeCabを生成します。

```
# 形態素解析のライブラリをインポート
import MeCab
import pandas as pd

# MeCabモデルを生成
tagger = MeCab.Tagger()
tagger
```

```
<Swig Object of type 'MeCab::Tagger *' at 0x7f7331093f10>
```

形態素解析の実行

それでは、MeCabのTaggerメソッドを使用して、形態素解析を行ってみましょう。

```
# 形態素解析を実行
node = tagger.parse("僕は美しい花が好きです")
print(node)
```

```
僕      名詞,代名詞,一般,*,*,*,僕,ボク,ボク
は      助詞,係助詞,*,*,*,*,は,ハ,ワ
美しい  形容詞,自立,*,*,形容詞・イ段,基本形,美しい,ウツクシイ,ウツクシイ
花      名詞,一般,*,*,*,*,花,ハナ,ハナ
が      助詞,格助詞,一般,*,*,*,が,ガ,ガ
好き    名詞,形容動詞語幹,*,*,*,*,好き,スキ,スキ
```

```
です　　助動詞,*,*,*,特殊・デス,基本形,です,デス,デス
EOS
```

補足: parseメソッドの戻り値は全部で10個です。
左から「表層形、品詞、品詞細分類1、品詞細分類2、品詞細分類3、活用型、活用形、原形、読み、発音」となっており、値が入っていない箇所はアスタリスク（*）になっています。

```
# 文章を変えて形態素解析を実行
node = tagger.parse("彼女はピコ太郎とYoutubeに出演したあとに恋ダンスを踊った。")
print(node)
```

```
彼女　　名詞,代名詞,一般,*,*,*,彼女,カノジョ,カノジョ
は　　助詞,係助詞,*,*,*,*,は,ハ,ワ
ピコ　　名詞,固有名詞,一般,*,*,*,ピコ,ピコ,ピコ
太郎　　名詞,固有名詞,人名,名,*,*,太郎,タロウ,タロー
と　　助詞,並立助詞,*,*,*,*,と,ト,ト
Youtube 名詞,一般,*,*,*,*,*
に　　助詞,格助詞,一般,*,*,*,に,ニ,ニ
出演　　名詞,サ変接続,*,*,*,*,出演,シュツエン,シュツエン
し　　動詞,自立,*,*,サ変・スル,連用形,する,シ,シ
た　　助動詞,*,*,*,特殊・タ,基本形,た,タ,タ
あと　　名詞,一般,*,*,*,*,あと,アト,アト
に　　助詞,格助詞,一般,*,*,*,に,ニ,ニ
恋　　名詞,一般,*,*,*,*,恋,コイ,コイ
ダンス　名詞,サ変接続,*,*,*,*,ダンス,ダンス,ダンス
を　　助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
踊っ　　動詞,自立,*,*,五段・ラ行,連用タ接続,踊る,オドッ,オドッ
た　　助動詞,*,*,*,特殊・タ,基本形,た,タ,タ
。　　記号,句点,*,*,*,*,。,。,。
EOS
```

注釈: 一見するとうまく処理ができているように見えますが、よく見ると「ピコ太郎」が「ピコ」と「太郎」に、また「恋ダンス」も「恋」と「ダンス」に分かれてしまいました。これはMeCabの辞書に「ピコ太郎」も「恋ダンス」も登録されていないからです。

新語辞書のインストール

mecab-ipadic-NEologd（以下、NEologd）は、MeCabと共に使う単語分かち書き辞書で、SNSやネット上の単語をもとに週2回以上更新され、新語・固有表現に強く、語彙数が多く、しかもオープンソース・ソフトウェアです。

新たにダウンロードしてNEologdをColaboratoryにインストールしましょう。

```
# MeCab用の外部辞書（NEologd）をインストールする
! git clone --depth 1 https://github.com/neologd/mecab-ipadic-neologd.git
```

```
! echo yes | mecab-ipadic-neologd/bin/install-mecab-ipadic-neologd -n -a
```

```
～ 出力省略 ～

[install-mecab-ipadic-NEologd] : Install completed.
[install-mecab-ipadic-NEologd] : When you use MeCab, you can set '/usr/lib/x86_64-
linux-gnu/mecab/dic/mecab-ipadic-neologd' as a value of '-d' option of MeCab.
[install-mecab-ipadic-NEologd] : Usage of mecab-ipadic-NEologd is here.
Usage:
    $ mecab -d /usr/lib/x86_64-linux-gnu/mecab/dic/mecab-ipadic-neologd ...

[install-mecab-ipadic-NEologd] : Finish..
[install-mecab-ipadic-NEologd] : Finish..
```

**注釈: ** この処理も1～2分ほどかかります。実行ボタンの進捗アイコン（グルグル）が止まるまで、しばらく待機しましょう。

NEologd辞書の呼び出し

インストールしたNEologdを使用して形態素解析をするための準備をします。

MeCab.Tagger() のオプションとしてmecab-ipadic-NEologdのパスを指定する必要があります。

Colaboratory上の辞書のパスは
echo `mecab-config --dicdir`"/mecab-ipadic-neologd"
で取得できます。

```
# パスを取得するのに必要なライブラリをインポート
import subprocess

# 辞書(mecab-ipadic-NEologd)のPathを取得
cmd='echo `mecab-config --dicdir`"/mecab-ipadic-neologd"'
path = (subprocess.Popen(cmd, stdout=subprocess.PIPE, shell=True).communicate()
[0]).decode('utf-8')

# MeCabの事前設定（辞書ファイルをオプションで指定）
tagger = MeCab.Tagger("-d {0}".format(path))
```

それではNEologd辞書を使用して、さきほど失敗した「ピコ太郎」「恋ダンス」の形態素解析に挑戦します。

```
node = tagger.parse("彼と彼女はピコ太郎とYoutubeに出演して恋ダンスを踊った。")
print(node)
```

```
彼　　名詞,代名詞,一般,*,*,*,彼,カレ,カレ
と　　助詞,格助詞,一般,*,*,*,と,ト,ト
彼女　名詞,代名詞,一般,*,*,*,彼女,カノジョ,カノジョ
は　　助詞,係助詞,*,*,*,*,は,ハ,ワ
ピコ太郎 名詞,固有名詞,人名,一般,*,*,ピコ太郎,ピコタロウ,ピコタロー
と　　助詞,並立助詞,*,*,*,*,と,ト,ト
Youtube 名詞,一般,*,*,*,*,*
に　　助詞,格助詞,一般,*,*,*,に,ニ,ニ
出演　名詞,サ変接続,*,*,*,*,出演,シュツエン,シュツエン
し　　動詞,自立,*,*,サ変・スル,連用形,する,シ,シ
て　　助詞,接続助詞,*,*,*,*,て,テ,テ
恋ダンス 名詞,固有名詞,一般,*,*,*,恋ダンス,コイダンス,コイダンス
を　　助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
踊っ　動詞,自立,*,*,五段・ラ行,連用タ接続,踊る,オドッ,オドッ
た　　助動詞,*,*,*,特殊・タ,基本形,た,タ,タ
。　　記号,句点,*,*,*,*,。,。,。
EOS
```

注釈: 無事に「ピコ太郎」「恋ダンス」が分解できました。

ストップワードの除去

形態素解析の結果を見ると、「と」「は」「に」「を」など、単語としての意味を持たない言葉も抽出されています。

このままでは単語の意味を把握する際にノイズとなってしまいますので、一度、これらを除去してみます。

こういった出現回数が多く、処理対象とすべきではない言葉を「ストップワード」と呼びます。

```
node = tagger.parseToNode("彼と彼女はピコ太郎とYoutubeに出演して恋ダンスを踊った。")

# 結果を格納するためのリストを作成
result = []

# 形態素解析の結果から一つずつ取り出して、品詞を確認するループ
while node is not None:
    # 品詞情報取得
    hinshi = node.feature.split(",")[0]
    if  hinshi in ["名詞"]:
        # 表層形の取得
        result.append(node.surface)
    elif hinshi in ["動詞", "形容詞"]:
        # 形態素情報から原形情報を取得
        result.append(node.feature.split(",")[6])
    node = node.next

result
```

```
['彼', '彼女', 'ピコ太郎', 'Youtube', '出演', 'する', '恋ダンス', '踊る']
```

注釈: 無事に単語だけを取り出せました。なお、少し専門的になりますが、node.surfaceでは単語の「表層形」が、node.feature.split(",")[6]では単語の「原形」を取得することができます。興味のある方は、プログラムを改変して確認してみてください。

ネガポジ判定

ネガポジ判定は、「感情分析」（Sentiment Analysis）と呼ばれる手法の一種で、発言や発想などが、前向き（ポジティブ）か後ろ向き（ネガティブ）かを判定することです。

ネガポジ判定を行う方法として、ポジティブかネガティブかを判定する辞書を作成して、それをもとにポジティブ度合い（ネガティブ度合い）を算出する方法が考えられます。

ただし、辞書を一から作成するのは大変ですので、ここでは東京工業大学の大村先生が作成された「単語感情極性対応表」を使用させていただきます。

出典：単語感情極性対応表（東京工業大学 高村大也先生）

『 感情極性とは、その語が一般的に良い印象を持つか（positive） 悪い印象を持つか（negative）を表した二値属性です。 例えば、「良い」、「美しい」などはpositiveな極性、 「悪い」、「汚い」などはnegativeな極性を持ちます。（中略）感情極性値は、語彙ネットワークを利用して自動的に計算されたものです。 もともと二値属性ですが、-1から+1の実数値を割り当てました。 -1に近いほどnegative、+1に近いほどpositiveと考えられます。』（東京工業大学 高村大也先生の説明より）

```
# 単語感情極性対応表を読み込む
df_nega_posi=
pd.read_csv('http://www.lr.pi.titech.ac.jp/~takamura/pubs/pn_ja.dic',
sep=':',encoding="SHIFT-JIS")
df_nega_posi.head(10)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	優れる	すぐれる	動詞	1
0	良い	よい	形容詞	0.999995
1	喜ぶ	よろこぶ	動詞	0.999979

	優れる	すぐれる	動詞	1
2	褒める	ほめる	動詞	0.999979
3	めでたい	めでたい	形容詞	0.999645
4	賢い	かしこい	形容詞	0.999486
5	善い	いい	形容詞	0.999314
6	適す	てきす	動詞	0.999295
7	天晴	あっぱれ	名詞	0.999267
8	祝う	いわう	動詞	0.999122
9	功績	こうせき	名詞	0.999104

注釈: 読み込めましたが、1行目「優れる　すぐれる　動詞　1」が、DataFrameの列名として扱われてしまいました。

```
# DataFrameに列名（columuns）を付加
df_nega_posi.columns = ['単語','読み','品詞', 'スコア']
df_nega_posi.head(10)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	単語	読み	品詞	スコア
0	良い	よい	形容詞	0.999995
1	喜ぶ	よろこぶ	動詞	0.999979
2	褒める	ほめる	動詞	0.999979
3	めでたい	めでたい	形容詞	0.999645
4	賢い	かしこい	形容詞	0.999486
5	善い	いい	形容詞	0.999314
6	適す	てきす	動詞	0.999295
7	天晴	あっぱれ	名詞	0.999267
8	祝う	いわう	動詞	0.999122

	単語	読み	品詞	スコア
9	功績	こうせき	名詞	0.999104

注釈: 一見して修正されたように見えますが、よく見ると「優れる　すぐれる　動詞　1」がなくなってしまいました。これではよくありませんので、今度は列名を付けて、csvファイルを読み込みます。

```
# DataFrameに列名（columuns）を付加して、単語感情極性対応表を読み込む
df_nega_posi=
pd.read_csv('http://www.lr.pi.titech.ac.jp/~takamura/pubs/pn_ja.dic',
sep=':',encoding="SHIFT-JIS",names=('単語','読み','品詞', 'スコア'))
df_nega_posi.head(10)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	単語	読み	品詞	スコア
0	優れる	すぐれる	動詞	1.000000
1	良い	よい	形容詞	0.999995
2	喜ぶ	よろこぶ	動詞	0.999979
3	褒める	ほめる	動詞	0.999979
4	めでたい	めでたい	形容詞	0.999645
5	賢い	かしこい	形容詞	0.999486
6	善い	いい	形容詞	0.999314
7	適す	てきす	動詞	0.999295
8	天晴	あっぱれ	名詞	0.999267
9	祝う	いわう	動詞	0.999122

列名を含めて無事に読み込めましたので、データを確認してみましょう。

```
# データの行数、列数を出力
df_nega_posi.shape
```

```
(55125, 4)
```

```
# データの末尾10件を抽出
df_nega_posi.tail(10)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	単語	読み	品詞	スコア
55115	下手	へた	名詞	-0.999831
55116	卑しい	いやしい	形容詞	-0.999860
55117	ない	ない	形容詞	-0.999882
55118	浸ける	つける	動詞	-0.999947
55119	罵る	ののしる	動詞	-0.999961
55120	ない	ない	助動詞	-0.999997
55121	酷い	ひどい	形容詞	-0.999997
55122	病気	びょうき	名詞	-0.999998
55123	死ぬ	しぬ	動詞	-0.999999
55124	悪い	わるい	形容詞	-1.000000

```
# スコアを条件にしてデータを抽出
df_nega_posi.query('スコア >= 0.50 & スコア <= 0.55')
```

注釈: データが74件出力されましたが、Jupyterの制限で中略されてしまいました。この制限を外すためには、pandasのdisplay.max_rowsというオプションを指定します。

```
# DataFrameの表示件数を100に設定する
pd.set_option('display.max_rows', 100)
df_nega_posi.query('スコア >= 0.50 & スコア <= 0.55')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	単語	読み	品詞	スコア
2472	怜悧	れいり	名詞	0.549718
2473	世にも	よにも	副詞	0.549163
2474	利達	りたつ	名詞	0.548724
2475	聖女	せいじょ	名詞	0.547436
2476	鍛冶	たんや	名詞	0.547367
2477	利発	りはつ	名詞	0.546507
2478	突入	とつにゅう	名詞	0.546454
2479	微笑	びしょう	名詞	0.545033
2480	鍛工	たんこう	名詞	0.544722
2481	適法	てきほう	名詞	0.544605
2482	佳人	かじん	名詞	0.543288
2483	愛育	あいいく	名詞	0.541709
2484	希代	きたい	名詞	0.541515
2485	真裸	まっぱだか	名詞	0.541345
2486	壮麗	そうれい	名詞	0.539643
2487	凛と	りんと	副詞	0.538857
2488	博雅	はくが	名詞	0.538053
2489	親昵	しんじつ	名詞	0.537723
2490	しっこし	しっこし	名詞	0.537091
2491	了知	りょうち	名詞	0.535344
2492	子煩悩	こぼんのう	名詞	0.535308
2493	酒盛	さかもり	名詞	0.533395
2494	途端	とたん	名詞	0.532354
2495	気前	きまえ	名詞	0.530800

	単語	読み	品詞	スコア
2496	達意	たつい	名詞	0.529638
2497	手負	ておい	名詞	0.529471
2498	作物	さくぶつ	名詞	0.529421
2499	麗人	れいじん	名詞	0.528980
2500	慈しむ	いつくしむ	動詞	0.526972
2501	穏和	おんわ	名詞	0.526812
2502	撫育	ぶいく	名詞	0.525700
2503	今夕	こんせき	名詞	0.525090
2504	奇態	きたい	名詞	0.525050
2505	小力	こぢから	名詞	0.524486
2506	稀覯	きこう	名詞	0.523364
2507	赴く	おもむく	動詞	0.522963
2508	未見	みけん	名詞	0.522578
2509	高め	たかめ	名詞	0.522058
2510	否応	いやおう	名詞	0.522003
2511	気散じ	きさんじ	名詞	0.521793
2512	気性	きしょう	名詞	0.521343
2513	言笑	げんしょう	名詞	0.521004
2514	勇士	ゆうし	名詞	0.519647
2515	栄耀	えいよう	名詞	0.517855
2516	御筆先	おふでさき	名詞	0.517826
2517	珍奇	ちんき	名詞	0.517630
2518	つと	つと	副詞	0.517366
2519	会心	かいしん	名詞	0.517358
2520	亀の子	かめのこ	名詞	0.517199
2521	換言	かんげん	名詞	0.516990
2522	有能	ゆうのう	名詞	0.516779
2523	美形	びけい	名詞	0.515789
2524	盲愛	もうあい	名詞	0.515466
2525	詳解	しょうかい	名詞	0.515385

	単語	読み	品詞	スコア
2526	能文	のうぶん	名詞	0.515224
2527	優女	やさおんな	名詞	0.514481
2528	壮絶	そうぜつ	名詞	0.514138
2529	塵埃	じんあい	名詞	0.513942
2530	今朝	けさ	名詞	0.512024
2531	早耳	はやみみ	名詞	0.511757
2532	真向い	まむかい	名詞	0.510080
2533	奇しくも	くしくも	副詞	0.509337
2534	寛大	かんだい	名詞	0.509142
2535	迸出	へいしゅつ	名詞	0.508903
2536	裸身	らしん	名詞	0.508858
2537	刀瘢	とうはん	名詞	0.508796
2538	気宇	きう	名詞	0.508739
2539	無上	むじょう	名詞	0.506431
2540	幽邃	ゆうすい	名詞	0.506234
2541	奉拝	ほうはい	名詞	0.505263
2542	澄し汁屋	すましや	名詞	0.504394
2543	きりきり	きりきりしゃんと	副詞	0.504162
2544	尊崇	そんすう	名詞	0.503112
2545	温和	おんわ	名詞	0.501493

それでは、任意の単語のネガポジ度を算出するプログラムを作成しましょう。まず、単語とスコアをワンセットとした辞書(dict)を登録します。

```
# 単語とスコアをワンセットとした辞書(dict)を作成
lst_tango = df_nega_posi['単語']
lst_score = df_nega_posi['スコア']
negaposi_dic   = dict(zip(lst_tango, lst_score))
negaposi_dic
```

```
{'優れる': 1.0,
 '良い': 0.9999950000000001,
 '喜ぶ': 0.9999790000000001,
 '褒める': 0.9999790000000001,
```

```
 'めでたい': 0.9996450000000001,
 '賢い': 0.9994860000000001,
 '善い': 0.9993139999999999,
 '適す': 0.999295,
 '天晴': 0.999267,
 '祝う': 0.999122,

 ～ 出力省略 ～

 ...}
```

それでは、上記で作成した辞書(negaposi_dic)に、ネガポジ度を算出したい単語を指定して実行しましょう。

```
# 任意の単語のネガポジ度を算出
negaposi_dic["危ない"]
```

```
-0.9967219999999999
```

せっかくですので、MeCabを使って、文章を形態素解析してから、ネガポジ判定をしてみましょう。

```
node = tagger.parseToNode("誕生日にお祝いしてくれました。本当に嬉しい！　みんなずっと友達だよ")
# node = tagger.parseToNode("昨日から調子が悪くて、病院に行ったら、病気が見つかりました。しばらく休みたいと思います")

# 結果を格納するためのリストを作成
result = []

# 形態素解析の結果から一つずつ取り出して、品詞を確認するループ
while node is not None:
    # 品詞情報取得
    hinshi = node.feature.split(",")[0]
    if  hinshi in ["名詞"]:
        # 表層形の取得
        result.append(node.surface)
    elif hinshi in ["動詞", "形容詞"]:
        # 形態素情報から原形情報を取得
        result.append(node.feature.split(",")[6])
    node = node.next

result
```

```
['誕生日', 'お祝い', 'する', 'くれる', '嬉しい', 'みんな', '友達']
```

```
# 形態素解析された結果を一つずつネガポジ判定するループ
for tango in result:

    if tango in negaposi_dic:
        pn_score = negaposi_dic[tango]
    else:
        pn_score = '---'

    display(str(pn_score)  + ":" + tango)
```

```
'---:誕生日'


'---:お祝い'


'---:する'


'---:くれる'


'0.998871:嬉しい'


'---:みんな'


'0.9605030000000001:友達'
```

以上で形態素解析とネガポジ判定の演習は終了です。これを発展させて、例えば、インターネット上の商品レビューや、SNSのコメントを収集して、ネガティブなのかポジティブなのかの指標を出すようなプログラムを作成してみるのも面白いでしょう。

単語をベクトル化する

単語をベクトル化することで、単語の持つ意味を数式でモデル化が可能になるのでは？という発想でGensimライブラリのword2vecモデルがあります。

膨大な自然言語データ（コーパス）を解析し、対象単語の周辺にある単語は、関係性が強いとすることで、独自の言語ベクトルを構築して、例えば、単語間の類似性や計算が可能になります。

- ”国王”－”男”＋”女”＝「女王」
- “イチロー”ー”野球”＋”サッカー”＝「女王」

```
import google.colab.drive
google.colab.drive.mount('gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?
client_id=947318989803-
6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aiet
f%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.goog
leapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20
https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fww
w.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at gdrive
```

```
# gensimライブラリからword2vecをインポート
from gensim.models import word2vec
```

さて、word2vecに用いる学習済みモデルをご紹介します。

これは、Wikipedia日本語の全データ（約3.7GB）をダウンロードして、そこからXMLタグを除去してMecabで形態素解析（解析時間は約30分）。

その後、Word2Vecのモデルに成形（i7のPCを使用して生成時間は約150分）したもので、ファイル名は「wiki.model」です

```
# Wikipediaを元に構築したモデルを読み込む（パスは皆さんの環境に書き換えてください）
model = word2vec.Word2Vec.load(r"/content/gdrive/My Drive/wiki/wiki.model")
```

```
/usr/local/lib/python3.6/dist-packages/smart_open/smart_open_lib.py:402:
UserWarning: This function is deprecated, use smart_open.open instead. See the
```

```
migration notes for details: https://github.com/RaRe-
Technologies/smart_open/blob/master/README.rst#migrating-to-the-new-open-function
  'See the migration notes for details: %s' % _MIGRATION_NOTES_URL
```

さっそく、word2vecによってベクトル化された単語で、様々な処理をしてみましょう。

まずは、ある単語に近い意味を持つ（であろう単語）を、類似度とともに出力してみます。

```
# ２つの単語を入力して、その単語の類似度を調べる
model.wv.similarity('イチロー','松井秀喜')
```

```
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning:
Conversion of the second argument of issubdtype from `int` to `np.signedinteger`
is deprecated. In future, it will be treated as `np.int64 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):




0.909052
```

次に、ベクトルの概念を使用して、単語の足し算をしてみましょう。

```
# positiveに足す単語、negativeに引く単語を入力する
results = model.wv.most_similar(positive=['国王','女'], negative=['男'])

for result in results:
    print(result)
```

```
('王太子', 0.8083631992340088)
('王妃', 0.8058691024780273)
('王室', 0.7935012578964233)
('女王', 0.7932999134063721)
('セーイソ', 0.7898644208908081)
('ワンチュク', 0.7862237691879272)
('皇帝', 0.7817065715789795)
('ノルウェー国王', 0.7781429290771484)
('コンスタンチン・パヴロヴィチ', 0.7737889289855957)
('王家', 0.7725479602813721)


/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning:
Conversion of the second argument of issubdtype from `int` to `np.signedinteger`
```

```
is deprecated. In future, it will be treated as `np.int64 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

```
results = model.wv.most_similar(positive=['イチロー','サッカー'], negative=['野
球'])

for result in results:
    print(result)
```

```
('リオネル・メッシ', 0.8135743141174316)
('中山雅史', 0.808011531829834)
('中田英寿', 0.7859162092208862)
('本田圭佑', 0.7841384410858154)
('ロナウジーニョ', 0.782444179058075)
('三浦知良', 0.7785511612892151)
('メッシ', 0.7772841453552246)
('クリスティアーノ・ロナウド', 0.7740367650985718)
('ディエゴ・マラドーナ', 0.7732782363891602)
('小野伸二', 0.7729653716087341)


/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning:
Conversion of the second argument of issubdtype from `int` to `np.signedinteger`
is deprecated. In future, it will be treated as `np.int64 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

word2vecを使用することで、単語の数値化が可能になりました。

このベクトルは特徴量になりえますので、ここからニューラルネットワークを用いることで、スパムメール分類モデルなど、より実用的な使用方法が考えられます。

AI2-5

AI の実装（画像認識）

－ 講義内容 －

- 画像認識のタスク
- データセットと手法
- 畳み込みニューラルネットワーク
- 演習：画像分類
- 演習：物体検出

AI 2-5
画像認識のタスクと手法

令和元年度厚生労働省　教育訓練プログラム開発事業　AI講座

2019年11月27日（水）PM

モリパワー株式会社　鬼島佳子

画像認識のタスク

画像認識のタスク

- 分類（Classification）
- 物体検出（Detection）
- セグメンテーション（Segmentation）
- 姿勢推定（Human Pose Estimation）　etc...

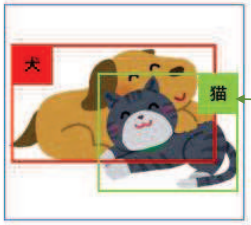
分類（Classification）

画像に何が写っているかを予測（１つの出力）

物体検出（Detection）

画像のどこに何が写っているかを予測（位置とクラス）

物体検出



「バウンディングボックス」
…物体を囲む矩形

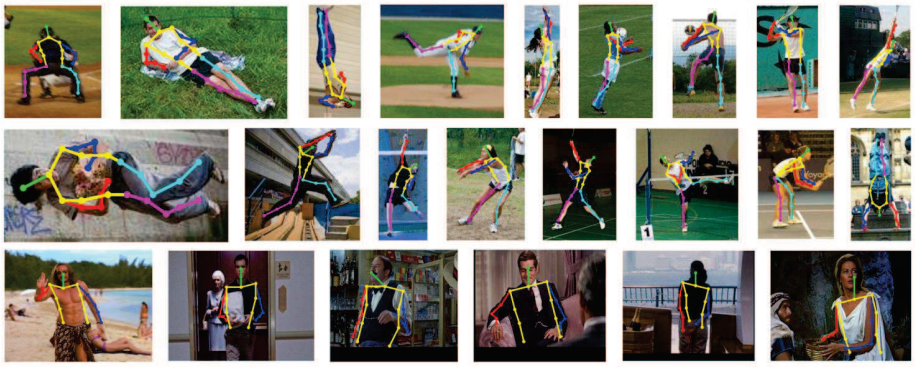
セグメンテーション（Segmentation）

ピクセル単位で、どの領域に属しているかを予測

- セマンティック・セグメンテーション（Semantic Segmentation）
  … ピクセル単位で、どのクラスに属するか予測
- インスタンス・セグメンテーション（Instance Segmentation）
  … ピクセル単位で、何のクラスの、どの個体に属するか予測

姿勢推定（Human Pose Estimation）

人間の関節の位置を予測



出典：https://nanonets.com/blog/human-pose-estimation-2d-guide/
日本語訳：https://qiita.com/KYoshiyama/items/9f5f5a13f957e138380b

データセットと手法

分類のデータセット

よくベンチマークに使われるもの

- MNIST　http://yann.lecun.com/exdb/mnist/
  手書き数字　10クラス　7万枚
  32px×32pxモノクロ
- CIFAR-10　https://www.cs.toronto.edu/~kriz/cifar.html
  10クラス　6,000枚
  32px×32pxカラー
- ImageNet　http://www.image-net.org/
  2万クラス　1,400万枚の画像のURL

分類の手法

畳み込みニューラルネットワーク（CNN）

有名なCNNアーキテクチャ

- AlexNet（2012）… 画像認識に初めて深層学習を利用した。深さ8層
- ResNet（2015）… 勾配消失問題を回避する残差学習の導入。深さ152層
- NasNet（2016）… モデル構造の最適化を行う。
- MobileNet（2017-2019）… 軽量なモデル。

各フレームワークの訓練済みモデル

- TensorFlow
  https://www.tensorflow.org/api_docs/python/tf/keras/applications
- PyTorch
  https://pytorch.org/docs/stable/torchvision/models.html

用途、求める精度、コスト（学習時間）、
実装されているフレームワークなどから使用するアーキテクチャを選定する

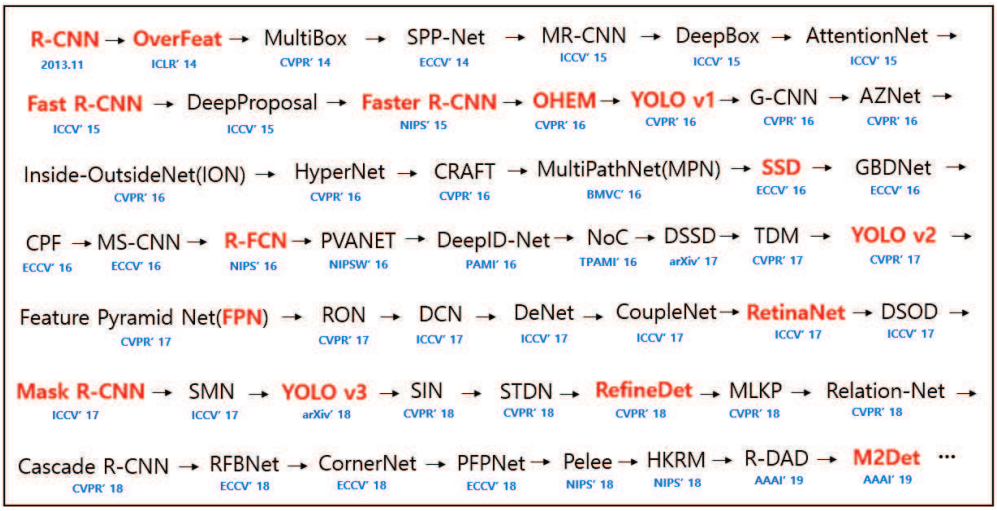
物体検出のデータセット

- COCO　http://cocodataset.org/
  80クラス　33万枚
  バウンディングボックス、セグメンテーション、キーポイント、キャプション（説明文）のアノテーション
- PASCAL VOC2012　http://host.robots.ox.ac.uk/pascal/VOC/voc2012/
  コンペで使用されたデータセット
  バウンディングボックス、セグメンテーション、行動や人間のパーツのアノテーション
- Open Images Dataset　https://storage.googleapis.com/openimages/web/
  Googleが公開している世界最大のデータセット
  画像URLに、バウンディングボックス、セグメンテーション、関連性のアノテーション

物体検出の手法

- R-CNN（Region-based CNN）
  - R-CNN [2013] / Fast R-CNN [2015] / Faster R-CNN [2015]
- YOLO（You Only Look Once）
  - YOLO [2015] / YOLOv2 [2016] / YOLOv3 [2018]
    https://pjreddie.com/darknet/yolo/
- SSD（Single Shot Detector）[2016]
  https://github.com/weiliu89/caffe/tree/ssd
- RefineDet [2017]
  https://github.com/sfzhang15/RefineDet
- M2Det [2018]
  https://github.com/qijiezhao/M2Det

物体検出の手法の変遷



出典：https://github.com/hoya012/deep_learning_object_detection

セグメンテーションのデータセット

- COCO
- PASCAL VOC2012
- OpenImagesDataset
- LVIS　https://www.lvisdataset.org/
  インスタンス・セグメンテーションのデータセット
  COCO2017の拡張

セグメンテーションの手法

セマンティック・セグメンテーション

- FCN（Fully Convolutional Network）[2015]
- U-Net[2015]
- SegNet[2015]
- PSPNet[2017]

インスタンス・セグメンテーション

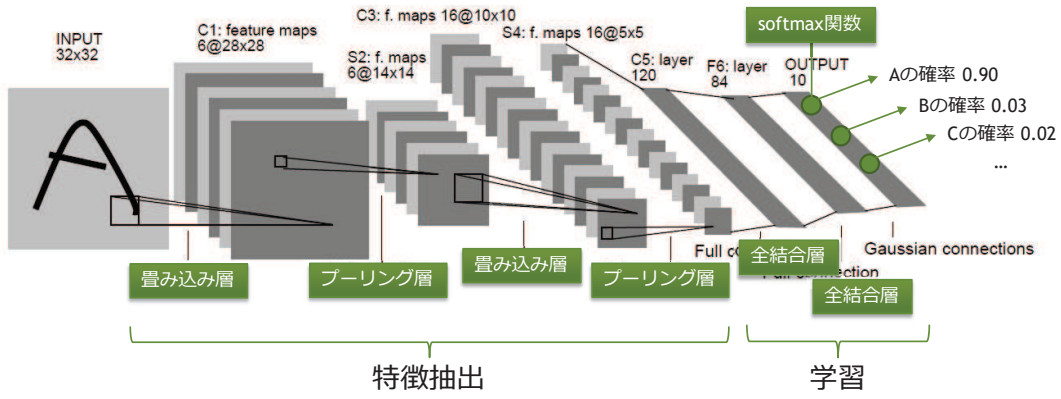
- Mask R-CNN[2017]
  https://github.com/matterport/Mask_RCNN

姿勢推定のデータセット

- COCO
- MPII Human Pose Dataset
  http://human-pose.mpi-inf.mpg.de/
- Leeds Sports Pose Dataset
  https://sam.johnson.io/research/lsp.html
  スポーツの画像のデータセット
- Frames Labeled In Cinema (FLIC)
  https://bensapp.github.io/flic-dataset.html
  映画のシーンのデータセット

畳み込みニューラルネットワーク
Convolutional Neural Networks:CNN

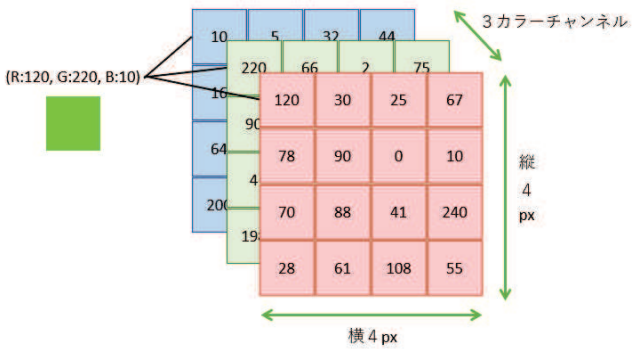
畳み込みニューラルネットワークの基本構造



出典：Gradient-Based Learning Applied to Document. Recognition. Yann LeCun, L eon Bottou, Yoshua Bengio, and Patrick Haffner

画像データ

画像データは色情報のあつまり

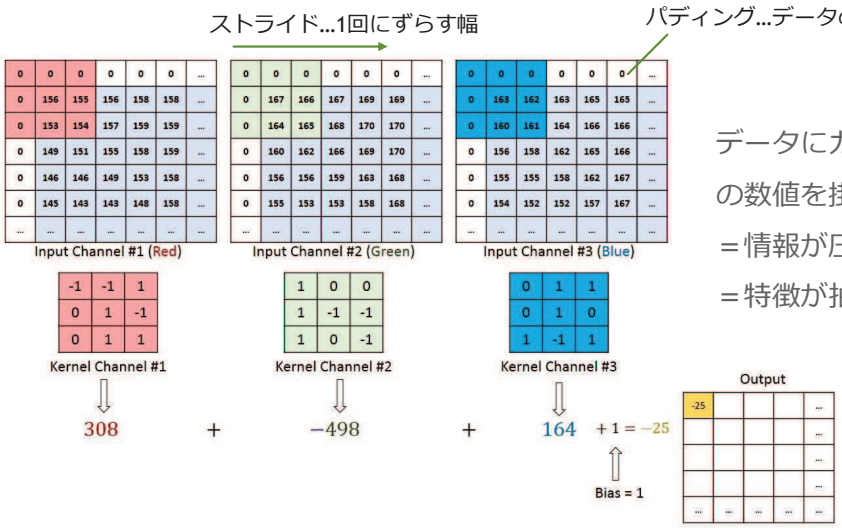


- 色は0～255の数値
- チャンネル数
  - カラー画像の場合
    Red, Green, Blueの３チャンネル
  - カラー（透明度あり）の場合
    Red, Green, Blue, Alphaの４チャンネル
  - モノクロ画像の場合
    黒の１チャンネル

４px×４pxのRGB画像なら

４×４×３＝48個の数値で表される

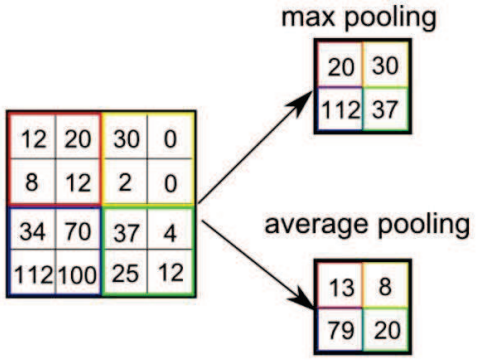
畳み込み



データにカーネル（フィルタ）
の数値を掛けて総和をとる
＝情報が圧縮される
＝特徴が抽出される

出典：https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

プーリング



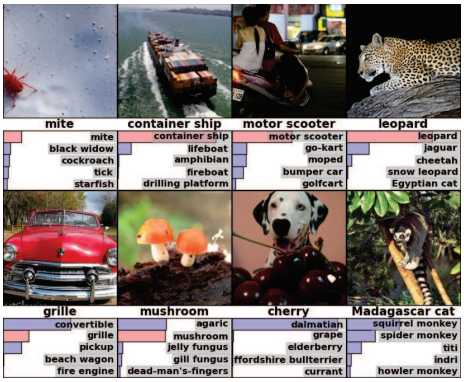
- Maxプーリング
  … 総和をとる
- Averageプーリング
  … 平均をとる

画像処理では特徴をより際立たせたいので、
Maxプーリングを使うことが多い

出典：https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

ソフトマックス関数

- それぞれのクラスに属する確率の総和が１（100%）になるように正規化する関数



出典：https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

ご清聴ありがとうございました

画像分類

CIFAR-10データセットを使って、何が映っている画像か分類してみましょう。
CIFAR-10は、ILVRC2012で優勝したAlex-netの名前の由来にもなったヒントン大学のAlex Krizhevsky氏が作成した画像データセットです。
データセットには、32px×32pxサイズ、RGBカラーの6万枚の画像が含まれており、飛行機や自動車など10種類のクラスに分類されています。

データセットの公式Web：
https://www.cs.toronto.edu/~kriz/cifar.html

ここで学習すること

- 画像データの仕組み
- データ拡張（Data Augmentation）
- 畳み込みニューラルネットワーク（CNN）の実装方法
- 学習中のチェックポイントの作成方法と、学習を途中から再開する方法
- オリジナルのデータセットの作り方

ライブラリのインポート

プログラムで使用するライブラリをインポートします。
今回は以下のライブラリを使用します。

- numpy … NumPy(https://numpy.org/) 数値計算に使用します。
- matplotlib … matplotlib(https://matplotlib.org/) グラフの描画に使用します。
- tensorflow … TensorFlow(https://www.tensorflow.org/?hl=ja) ニューラルネットワークを使用するためのフレームワークです。
- tensorflow.keras … Keras(https://keras.io/ja/) TensorFlowのラッパーです。
- sklearn … scikit-learn(https://scikit-learn.org/stable/) 機械学習のライブラリです。

```
# matplotlibの最新バージョンを使用する
!pip install matplotlib --upgrade
```

```
Collecting matplotlib
□[?25l  Downloading
https://files.pythonhosted.org/packages/4e/11/06958a2b895a3853206dea1fb2a5b11bf044
f626f90745987612af9c8f2c/matplotlib-3.1.2-cp36-cp36m-manylinux1_x86_64.whl
(13.1MB)
□[K     |████████████████████████████████| 13.1MB 4.8MB/s
□[?25hRequirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in
/usr/local/lib/python3.6/dist-packages (from matplotlib) (1.1.0)
Requirement already satisfied, skipping upgrade: cycler>=0.10 in
/usr/local/lib/python3.6/dist-packages (from matplotlib) (0.10.0)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 in
```

```
/usr/local/lib/python3.6/dist-packages (from matplotlib) (2.6.1)
Requirement already satisfied, skipping upgrade: numpy>=1.11 in
/usr/local/lib/python3.6/dist-packages (from matplotlib) (1.17.4)
Requirement already satisfied, skipping upgrade:
pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages
(from matplotlib) (2.4.5)
Requirement already satisfied, skipping upgrade: setuptools in
/usr/local/lib/python3.6/dist-packages (from kiwisolver>=1.0.1->matplotlib)
(41.6.0)
Requirement already satisfied, skipping upgrade: six in
/usr/local/lib/python3.6/dist-packages (from cycler>=0.10->matplotlib) (1.12.0)
□[31mERROR: albumentations 0.1.12 has requirement imgaug<0.2.7,>=0.2.5, but
you'll have imgaug 0.2.9 which is incompatible.□[0m
Installing collected packages: matplotlib
  Found existing installation: matplotlib 3.1.1
    Uninstalling matplotlib-3.1.1:
      Successfully uninstalled matplotlib-3.1.1
Successfully installed matplotlib-3.1.2
```

```
# ライブラリのインポート
import numpy as np
import matplotlib.pyplot as plt

# TensorFlow 2.x系を使う
%tensorflow_version 2.x
import tensorflow as tf
from tensorflow.keras import models, layers, regularizers
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint

# TensorFlowのバージョン
print(tf.__version__)

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

# matplotlib の出力をノートブック上に描画するための設定
%matplotlib inline
```

```
TensorFlow 2.x selected.
2.0.0
```

データの読み込み

CIFAR-10データセットを読み込みましょう。
Kerasの `datasets.cifar10.load_data()` を使ってデータを読み込みます。

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 4s 0us/step
```

データの分析

読み込んだデータセットの内容を確認しましょう。

```
print('データ型')
print('x_train type:', type(x_train), x_train.dtype)
print('y_train type:', type(y_train), y_train.dtype)
print('x_test type:', type(x_test), x_test.dtype)
print('y_test type:', type(y_test), y_test.dtype)

print('\n行数, 列数')
print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)
print('x_test shape:', x_test.shape)
print('y_test shape:', y_test.shape)
```

```
データ型
x_train type: <class 'numpy.ndarray'> uint8
y_train type: <class 'numpy.ndarray'> uint8
x_test type: <class 'numpy.ndarray'> uint8
y_test type: <class 'numpy.ndarray'> uint8

行数, 列数
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
x_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 1)
```

データ型はいずれもnumpy.ndarray（配列）、符号なし整数です。
訓練データは50,000件、評価データは10,000件あります。

画像データの仕組み

画像のデータは、ピクセルごとの色情報の集まりです。
データの形状は(32, 32, 3) でした。つまり、32ピクセル×32ピクセル分、RGBチャンネル（赤、緑、青の

3種類）の値が格納されています。
RGBはそれぞれ、0～255の値で表します。
データの形状は、色情報がRGBA（RGB+透明度）の場合は(32, 32, 4)、モノクロの場合は(32, 32, 1) になります。

では、1件目のデータの中身を見てみましょう。

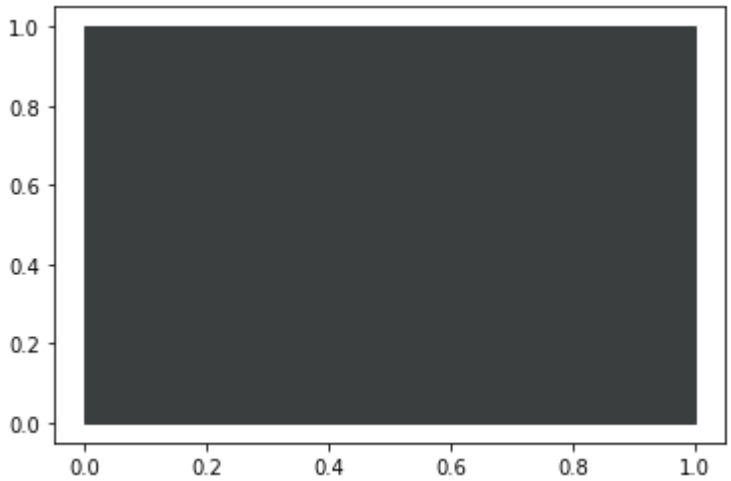
```
print(x_train[0])
```

```
[[[ 59  62  63]
  [ 43  46  45]
  [ 50  48  43]
  ...
  [158 132 108]
  [152 125 102]
  [148 124 103]]

 [[ 16  20  20]
  [  0   0   0]
  [ 18   8   0]
  ...
  [123  88  55]
  [119  83  50]
  [122  87  57]]

 [[ 25  24  21]
  [ 16   7   0]
  [ 49  27   8]
  ...
  [118  84  50]
  [120  84  50]
  [109  73  42]]

 ...

 [[208 170  96]
  [201 153  34]
  [198 161  26]
  ...
  [160 133  70]
  [ 56  31   7]
  [ 53  34  20]]

 [[180 139  96]
  [173 123  42]
  [186 144  30]
  ...
  [184 148  94]
```

```
  [ 97  62  34]
  [ 83  53  34]]

 [[177 144 116]
  [168 129  94]
  [179 142  87]
  ...
  [216 184 140]
  [151 118  84]
  [123  92  72]]]
```

最初の要素の [59, 62, 63] は、R（赤）が59、G（緑）が62、B（青）が63という意味です。
どんな色なのか、見てみましょう。

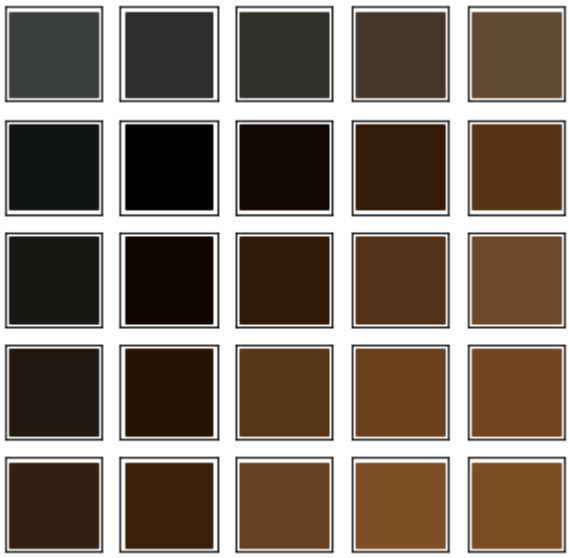
```
# matplotlibでは色は0〜1の範囲で指定する必要があるので、255で割る
color_r = 59 / 255  # 赤
color_g = 62 / 255  # 緑
color_b = 63 / 255  # 青

# 色で塗りつぶす
plt.fill([0,0,1,1], [0,1,1,0], color=(color_r, color_g, color_b))
plt.show()
```



画像の左上隅の5×5ピクセル分の色情報を描画してみると、以下のようになります。

```
y_num = 5
x_num = 5

# 5×5のサブプロット
f, axarr = plt.subplots(y_num, x_num, figsize=(x_num, y_num))

for i in range(x_num):
  for j in range(y_num):

    # 色情報
```

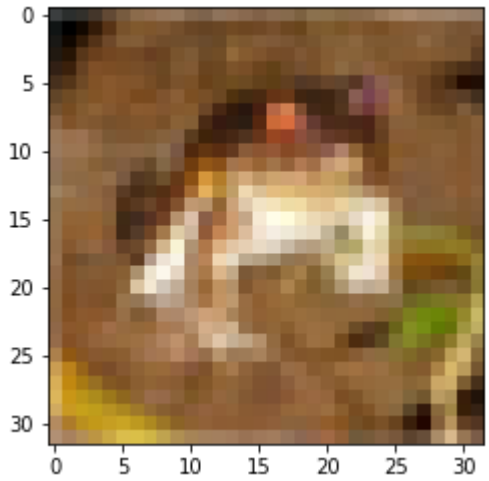
```
        color_r = x_train[0, j, i, 0] / 255  # 赤
        color_g = x_train[0, j, i, 1] / 255  # 緑
        color_b = x_train[0, j, i, 2] / 255  # 青

        # 塗りつぶし
        axarr[j, i].fill([0,0,1,1], [0,1,1,0], color=(color_r, color_g, color_b))

        # メモリを消す
        axarr[i, j].get_xaxis().set_visible(False)
        axarr[i, j].get_yaxis().set_visible(False)
```



このような色情報を32×32並べたものが画像です。

画像全体を見てみましょう。
画像は `plt.imshow()` を使って出力します。

```
    plt.imshow(x_train[0])
    plt.show()
```



ラベル

ラベルの内容も見ておきましょう。
このデータセットに用意されているクラスは10種類、ラベルにはクラスを表す数値が格納されています。

値	クラス名
0	airplane (飛行機)
1	automobile (自動車)
2	bird (鳥)
3	cat (猫)
4	deer (鹿)
5	dog (犬)
6	frog (カエル)
7	horse (馬)
8	ship (船)
9	truck (トラック)

数字では分かりづらいので、クラス名を定義しておきます。

```
# カテゴリ値とクラス名の対応付け
class_names = {0:'airplane', 1:'automobile', 2:'bird', 3:'cat', 4:'deer', 5:'dog',
6:'frog', 7:'horse', 8:'ship', 9:'truck'}

# クラス数
num_classes = len(class_names)  # 10
```

訓練データ1枚目のラベルを見てみましょう。

```
# 最初のデータに対応するラベル
y_train[0]
```

```
array([6], dtype=uint8)
```

1枚目の画像のラベルは「6」でした。
「6」に対応するクラス名を表示します。

```
# クラス名を表示
class_names[y_train[0, 0]]
```

```
'frog'
```

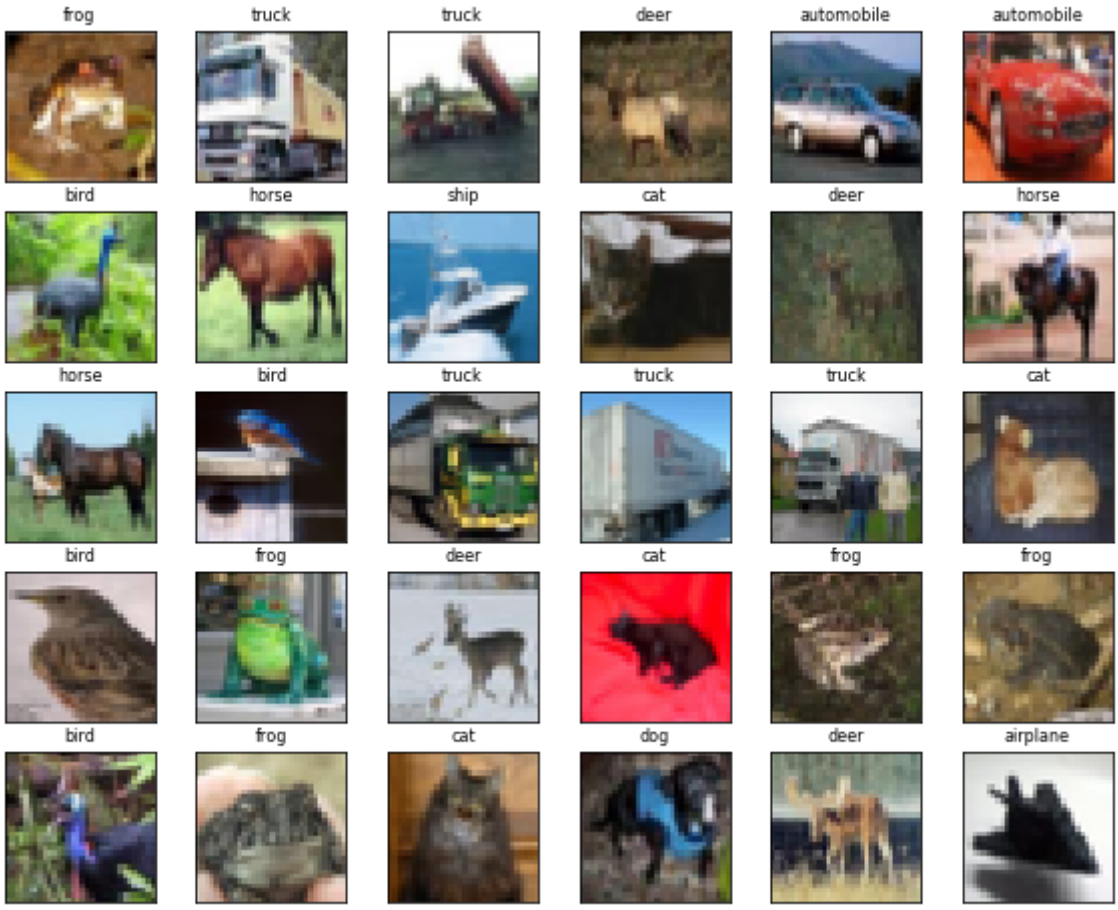
1枚目の画像には「カエル」のラベルがつけられていました。

他の画像も見てみましょう。

```
# 画像を並べて表示する関数
def show_img(x, y):
  plt.clf()

  # サイズ
  plt.figure(figsize=(10, 8))

  # 30枚の画像を表示
  for i in range(0, 30):
    plt.subplot(5, 6, i+1)

    # クラス名を表示
    plt.title(class_names[y[i]], fontsize=8)

    # 画像を表示
    fig = plt.imshow(x[i])

    # メモリを消す
    fig.axes.get_xaxis().set_visible(False)
    fig.axes.get_yaxis().set_visible(False)
```

```
show_img(x_train, y_train.flatten())
```

```
<Figure size 432x288 with 0 Axes>
```

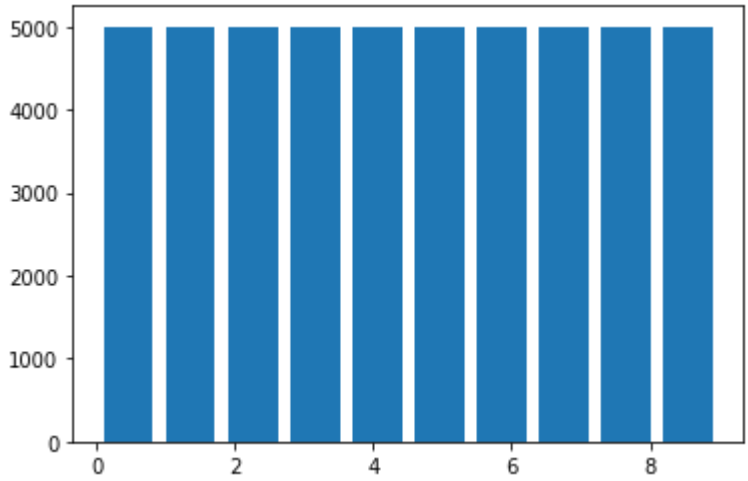


データの分布

訓練データのクラスの分布を確認しましょう。

```
plt.hist(y_train, bins=num_classes, rwidth=0.8)
```

```
(array([5000., 5000., 5000., 5000., 5000., 5000., 5000., 5000., 5000.,
        5000.]),
 array([0. , 0.9, 1.8, 2.7, 3.6, 4.5, 5.4, 6.3, 7.2, 8.1, 9. ]),
 <a list of 10 Patch objects>)
```



すべてのクラスが5,000件ずつ含まれており、偏りはありません。

前処理

正規化

各色の値は0～255でした。
このままでは値の範囲が広く学習に時間がかかるため、0～1の範囲に変換します。

```
# 0～1の範囲に変換するため、255で割る
x_train = x_train / 255.0
x_test = x_test / 255.0

print(x_train[0])
```

```
[[[0.23137255 0.24313725 0.24705882]
  [0.16862745 0.18039216 0.17647059]
  [0.19607843 0.18823529 0.16862745]
  ...
  [0.61960784 0.51764706 0.42352941]
  [0.59607843 0.49019608 0.4       ]
  [0.58039216 0.48627451 0.40392157]]

 [[0.0627451  0.07843137 0.07843137]
  [0.         0.         0.        ]
  [0.07058824 0.03137255 0.        ]
  ...
  [0.48235294 0.34509804 0.21568627]
  [0.46666667 0.3254902  0.19607843]
  [0.47843137 0.34117647 0.22352941]]

 [[0.09803922 0.09411765 0.08235294]
  [0.0627451  0.02745098 0.        ]
  [0.19215686 0.10588235 0.03137255]
  ...
```

```
  [0.4627451  0.32941176 0.19607843]
  [0.47058824 0.32941176 0.19607843]
  [0.42745098 0.28627451 0.16470588]]

 ...

 [[0.81568627 0.66666667 0.37647059]
  [0.78823529 0.6        0.13333333]
  [0.77647059 0.63137255 0.10196078]
  ...
  [0.62745098 0.52156863 0.2745098 ]
  [0.21960784 0.12156863 0.02745098]
  [0.20784314 0.13333333 0.07843137]]

 [[0.70588235 0.54509804 0.37647059]
  [0.67843137 0.48235294 0.16470588]
  [0.72941176 0.56470588 0.11764706]
  ...
  [0.72156863 0.58039216 0.36862745]
  [0.38039216 0.24313725 0.13333333]
  [0.3254902  0.20784314 0.13333333]]

 [[0.69411765 0.56470588 0.45490196]
  [0.65882353 0.50588235 0.36862745]
  [0.70196078 0.55686275 0.34117647]
  ...
  [0.84705882 0.72156863 0.54901961]
  [0.59215686 0.4627451  0.32941176]
  [0.48235294 0.36078431 0.28235294]]]
```

検証データの分割

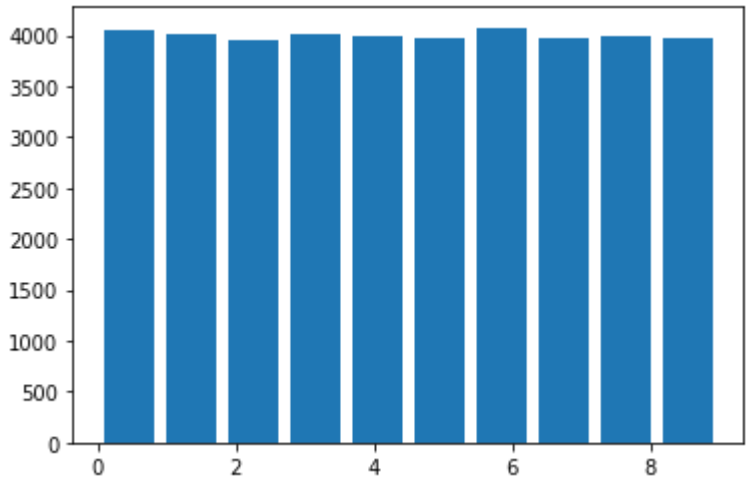
検証に使用するデータを分割しておきます。
訓練データのうち2割を検証データとします。

```
# 訓練データ8割、検証データ2割に分割
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2,
random_state=777)

print('訓練データ件数: ', len(x_train))
print('検証データ件数: ', len(x_val))
```

```
訓練データ件数:  40000
検証データ件数:  10000
```

```
# データの分布
plt.hist(y_train, bins=num_classes, rwidth=0.8)
```

```
(array([4050., 4016., 3958., 4010., 3993., 3969., 4075., 3976., 3984.,
        3969.]),
 array([0. , 0.9, 1.8, 2.7, 3.6, 4.5, 5.4, 6.3, 7.2, 8.1, 9. ]),
 <a list of 10 Patch objects>)
```



カテゴリ変数のone-hotベクトル化

ラベルには0～9のカテゴリ変数が格納されていました。
カテゴリ変数はそのままでは扱えませんので、one-hotベクトルに変換します。
one-hotベクトルとは、該当するクラスのインデックスのみ 1 、残りは 0 が格納されている配列です。

例）

[1,0,0,0,0,0,0,0,0,0] # 0:airplane
[0,1,0,0,0,0,0,0,0,0] # 1:automobile
[0,0,1,0,0,0,0,0,0,0] # 2:bird
…
[0,0,0,0,0,0,0,0,1,0] # 8:ship
[0,0,0,0,0,0,0,0,0,1] # 9:truck

`tf.keras.utils.to_categorical()` はカテゴリ変数をone-hotベクトルに変換する関数です。

```
# ラベルのone-hotベクトル化
y_train = tf.keras.utils.to_categorical(y_train, num_classes)  # 訓練データのラベル
y_val = tf.keras.utils.to_categorical(y_val, num_classes)      # 検証データのラベル
y_test = tf.keras.utils.to_categorical(y_test, num_classes)    # 評価データのラベル
```

```
print(y_train[:5])
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
```

データ拡張（Data Augmentation）

学習データとなる画像を少しずつ加工しながら水増しすることで、データ量の増加と汎化性能の向上が期待できます。
画像の水増しの手法はスタンフォード大学のチートシートによくまとまっています。

元の画像	反転	回転	ランダムな切り抜き
			
- 何も変更されていない画像	- 画像の意味が変わらない軸における反転	- わずかな角度の回転 - 不正確な水平線の校正（calibration）をシミュレートする	- 画像の一部へのランダムなフォーカス - 連続して数回のランダムな切り抜きが可能

カラーシフト	ノイズの付加	情報損失	コントラストの修正
			
- RGBのわずかな修正 - 照らされ方によるノイズを捉える	- ノイズの付加 - 入力画像の品質のばらつきへの耐性の強化	- 画像の一部を無視 - 画像の一部が欠ける可能性を再現する	- 明るさの変化 - 時刻による露出の違いをコントロールする

画像引用：https://github.com/afshinea/stanford-cs-230-deep-learning/blob/master/ja/cheatsheet-deep-learning-tips-tricks.pdf

Kerasにはデータ拡張のための機能 `ImageDataGenerator` が用意されています。
https://keras.io/ja/preprocessing/image/

回転やずらす割合などを指定すると、ランダムに画像を加工します。
加工には注意点があります。映っているものの「角度」や「向き」に意味がある場合は、回転や反転は行いません。

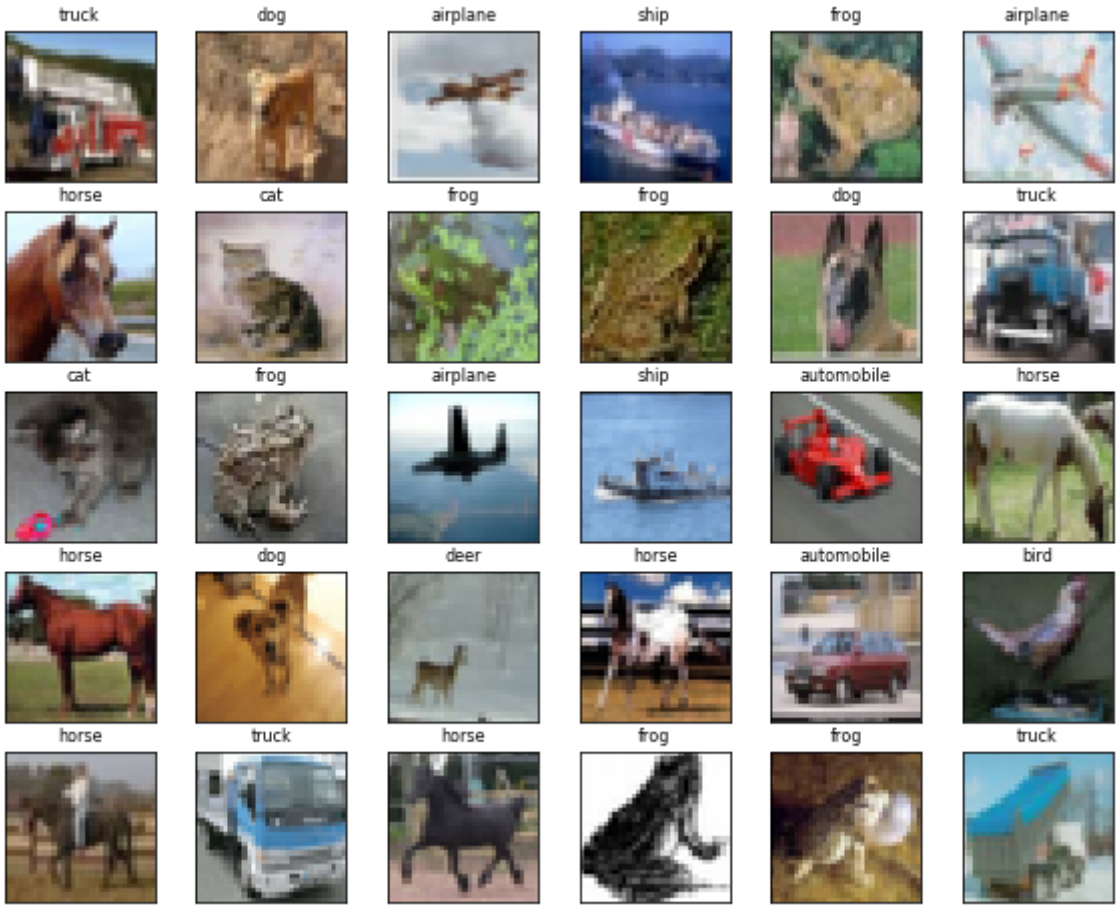
例えば上矢印と下矢印の画像を分類する場合、上下反転してしまうと意味が変わってしまいます。
同様に、色に意味がある場合は、色を大きく加工してはいけません（信号機の画像など）。

```
# バッチサイズを決める
batch_size = 64
```

```
# ImageDataGenerator
datagen = ImageDataGenerator(rotation_range=30,              # 回転する角度の範囲
                             width_shift_range=0.1,          # 横にずらす割合
                             height_shift_range=0.1,         # 縦にずらす割合
                             horizontal_flip=True,           # 水平に反転する
                             zoom_range=0.1,                 # ズームする割合
                             channel_shift_range=0.05)       # 各色に値を足す範囲

# 訓練データをデータ拡張
train_generator = datagen.flow(x_train, y_train, batch_size=batch_size,
shuffle=False)
```

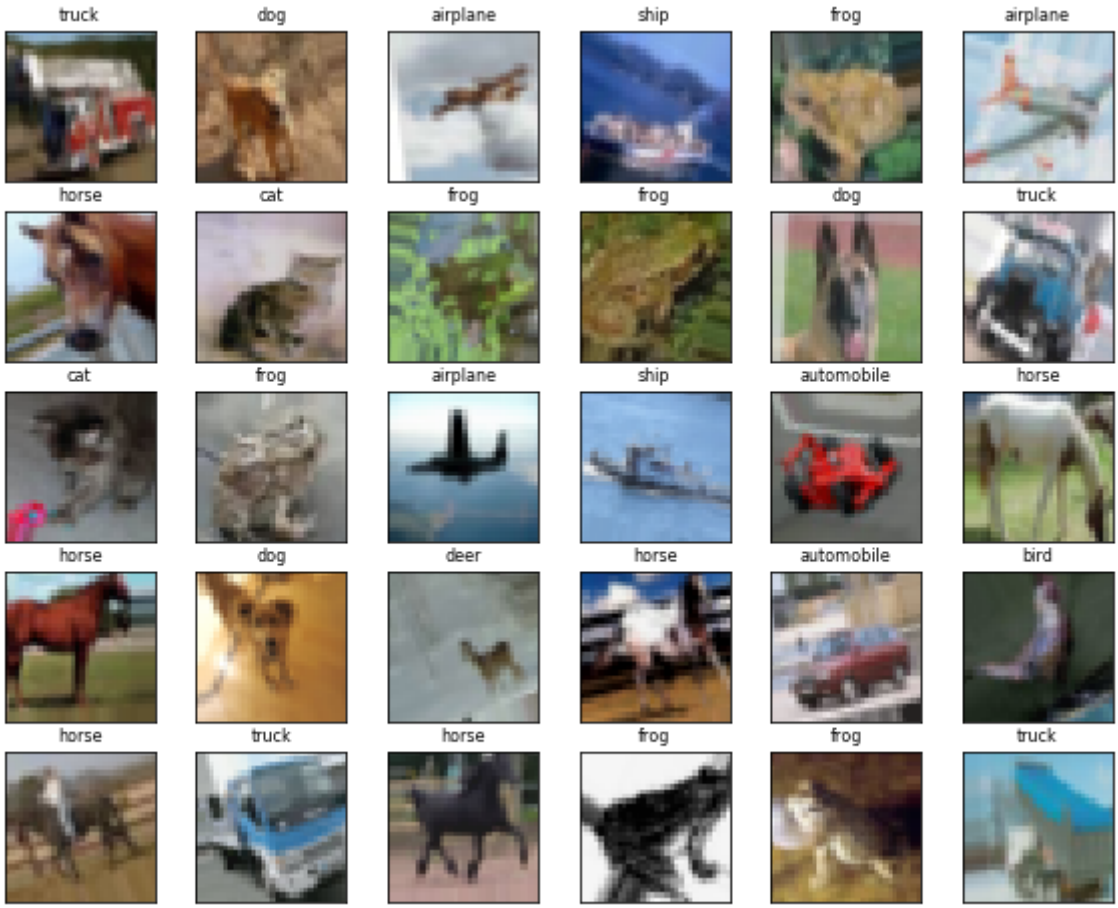
```
# 変更前の画像
show_img(x_train, [np.argmax(y) for y in y_train])
```

```
<Figure size 432x288 with 0 Axes>
```




```
# データ拡張した画像
x, y = next(train_generator)  # xがデータ拡張した画像、yがラベル
show_img(x, [np.argmax(yi) for yi in y])
```

```
<Figure size 432x288 with 0 Axes>
```



モデルの定義

画像を扱うタスクでは、畳み込みニューラルネットワークが有効とされています。

畳み込みニューラルネットワーク（Convolutional Neural Network：CNN）

畳み込みニューラルネットワークとは、「畳み込み層」と「プーリング層」を重ねたニューラルネットワークです。
畳み込み層では、小さなフィルタ（カーネル）を使って入力の特徴を抽出します。
プーリング層はMax（最大値）プーリングやAverage（平均）プーリングといった種類があり、情報の最大値や平均値を求めることで情報を集約します。
画像を扱う場合は特徴をより際立たせたいので、Maxプーリングを用いることが多いです。

隠れ層

畳み込み層 `layers.Conv2D` と、Maxプーリング層 `layers.MaxPool2D` をいくつか重ねます。
最初の層には入力の形状 `input_shape` を指定する必要がありますので、今回のデータの形状である (32, 32, 3) を指定します。

畳み込み層では、カーネルのサイズ `kernel_size` と、カーネルをどれだけずらすかを表すストライド `strides` を指定します。
また、パディング `padding` を指定することで、畳み込みによって画像が小さくなりすぎたり、画像の端の情報が反映されにくくなるのを防ぎます。

出力層

このモデルには最後に分類結果を出力させたいので、出力層には全結合層 `Dense` を使用します。
隠れ層の出力は3次元ですので、Dense 層に渡す前に、1次元に変換する（平坦化する）必要があります。
そこで、 Dense 層の前に、平坦化を行う `layers.Flatten` 層を挟みます。

最後の層の出力は、それぞれのクラスに該当する確率にしたいので、活性化関数は softmax を使用します。
今回は 10 クラスに分類しますので、出力層の最後の層のユニット数は 10 を指定します。

まずは小さな畳み込みニューラルネットワークを作成してみましょう。

```
# モデルの定義
model = tf.keras.Sequential([

    # --- 隠れ層 --- #

    # 畳み込み層
    layers.Conv2D(32, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu', input_shape=(32,32,3)),
    # Maxプーリング層
    layers.MaxPool2D(pool_size=(2,2), strides=(2,2)),

    # # 畳み込み層
    layers.Conv2D(64, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu'),
    # Maxプーリング層
    layers.MaxPool2D(pool_size=(2,2), strides=(2,2)),

    # 平坦化
    layers.Flatten(),

    # --- 出力層 --- #

    # 結合層
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes, activation='softmax'),

])

# モデルの構造を出力
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
```

```
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 128)               524416
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 545,098
Trainable params: 545,098
Non-trainable params: 0
_________________________________________________________________
```

モデルをコンパイルします。
カテゴリ分類ですので、損失関数は「categorical_crossentropy」を使用します。
評価関数は、どれくらい正しく分類できたかを示す「accuracy」を指定します。

```
# モデルのコンパイル
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

学習

学習途中でチェックポイントを保存する

機械学習には時間がかかる場合がありますので、学習を途中で中断したり、学習を途中から再開するやり方を覚えておきましょう。
Google Colaboratory は最大12時間連続で使用することができますが、時間内に学習が終わらな買った場合、保存していないモデルは削除されてしまいます。
そこで、学習の途中でモデルや学習済みの重みを保存しておき、次回学習する時は続きから再開できるようにします。

Googleドライブに保存しますので、Googleドライブをマウントします。

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?
client_id=947318989803-
6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aiet
f%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.goog
leapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20
```

```
https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fww
w.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

エポックごとにモデルを保存するための仕組みが、`ModelCheckpoint` です。
学習を行う `model.fit()` や `model.fit_generator()` の、`callbacks` オプションに指定することで、
毎エポック終了時にモデル（または重み）を保存してくれます。

```
# 保存先のパス
save_path = '/content/drive/My Drive/'

# 保存するファイル名
save_filename = 'cifar10_weight_{epoch:02d}.h5'

# チェックポイント
checkpoint = ModelCheckpoint(save_path + save_filename,
                             save_weights_only=True)  # 重みだけを保存
```

ImageDataGeneratorを使った学習

さて、いよいよ学習を行います。
ImageDataGenerator を利用して学習を行うには、 `model.fit_generator()` を使います。

```
# 学習とエポックごとの評価
history = model.fit_generator(train_generator,                    # 訓練データの
ImageDataGenerator
                    steps_per_epoch=(len(x_train)//batch_size),   # 1エポックの間に
generatorが生成するサンプルバッチ数（一般的にはデータ数÷バッチサイズ）
                    epochs=10,                                    # エポック数
                    validation_data=(x_val,y_val),                # 検証データ
                    callbacks=[checkpoint])                       # エポック終了時
に実行する処理（チェックポイントを保存）
```

```
Epoch 1/10
625/625 [==============================] - 43s 70ms/step - loss: 1.6757 -
accuracy: 0.3951 - val_loss: 1.4172 - val_accuracy: 0.4893
Epoch 2/10
625/625 [==============================] - 41s 66ms/step - loss: 1.4010 -
accuracy: 0.5044 - val_loss: 1.3342 - val_accuracy: 0.5204
Epoch 3/10
625/625 [==============================] - 41s 65ms/step - loss: 1.2957 -
accuracy: 0.5378 - val_loss: 1.1343 - val_accuracy: 0.5919
Epoch 4/10
```

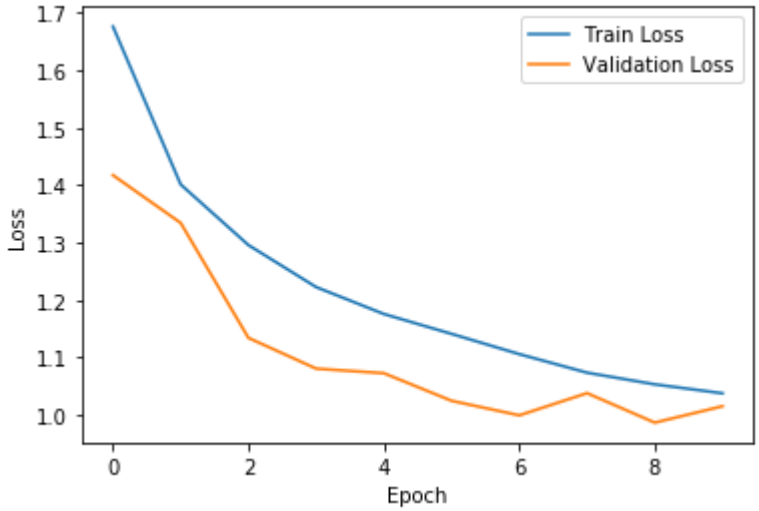
```
625/625 [==============================] - 41s 65ms/step - loss: 1.2226 -
accuracy: 0.5669 - val_loss: 1.0808 - val_accuracy: 0.6162
Epoch 5/10
625/625 [==============================] - 41s 65ms/step - loss: 1.1758 -
accuracy: 0.5830 - val_loss: 1.0729 - val_accuracy: 0.6312
Epoch 6/10
625/625 [==============================] - 40s 65ms/step - loss: 1.1412 -
accuracy: 0.5944 - val_loss: 1.0249 - val_accuracy: 0.6416
Epoch 7/10
625/625 [==============================] - 41s 65ms/step - loss: 1.1060 -
accuracy: 0.6083 - val_loss: 0.9996 - val_accuracy: 0.6467
Epoch 8/10
625/625 [==============================] - 41s 65ms/step - loss: 1.0738 -
accuracy: 0.6220 - val_loss: 1.0382 - val_accuracy: 0.6412
Epoch 9/10
625/625 [==============================] - 41s 66ms/step - loss: 1.0535 -
accuracy: 0.6289 - val_loss: 0.9869 - val_accuracy: 0.6550
Epoch 10/10
625/625 [==============================] - 41s 66ms/step - loss: 1.0379 -
accuracy: 0.6345 - val_loss: 1.0154 - val_accuracy: 0.6505
```

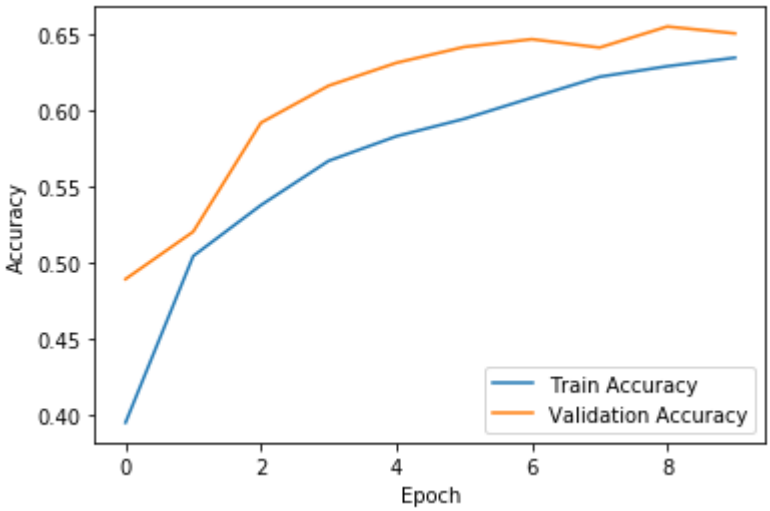
中断した学習を再開する場合は、 `model.load_weights()` で保存済みの重みファイルを読み込みます。

```
# # 保存したモデルの重みを読み込む
# model.load_weights(save_path + 'cifar10_weight_01.h5')

# # 続きから学習を再開
# history = model.fit_generator(train_generator,      # 訓練データの
ImageDataGenerator
#                     epochs=20,                      # エポック数
#                     validation_data=val_generator,  # 検証データの
ImageDataGenerator
#                     callbacks=[checkpoint])         # エポックごとにチェックポイン
ト保存
```

```
# 学習の推移をグラフ表示
# 損失
plt.figure()
plt.plot(history.epoch, history.history['loss'], label='Train Loss')
plt.plot(history.epoch, history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# 正解率
plt.figure()
plt.plot(history.epoch, history.history['accuracy'], label='Train Accuracy')
```

```
plt.plot(history.epoch, history.history['val_accuracy'], label='Validation
Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





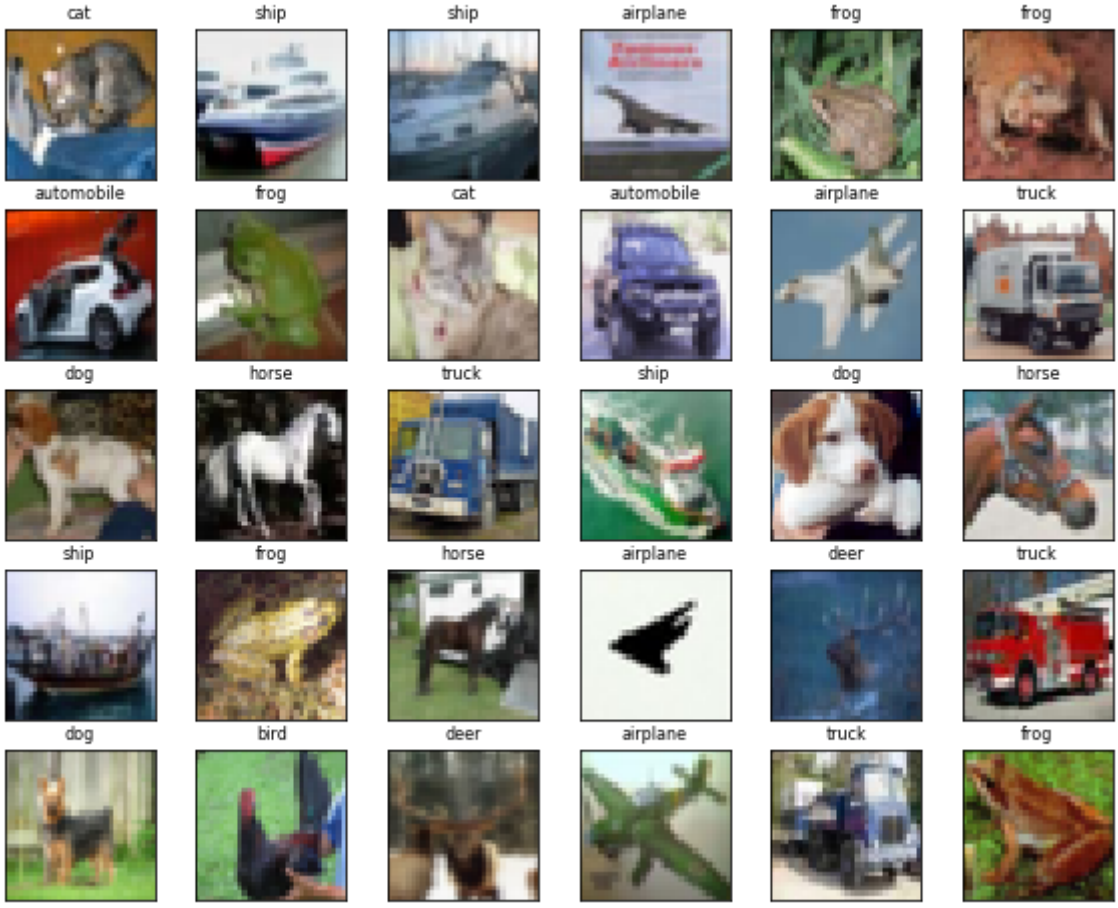
推論

テストデータを使って推論を行います。
先にテストデータを確認しておきましょう。

```
# テストデータのラベル
y_test_flat = [np.argmax(i) for i in y_test]
```

```
# 画像を表示
show_img(x_test, y_test_flat)
```

```
<Figure size 432x288 with 0 Axes>
```




```
# 推論
y_pred = model.predict(x_test)
```

`model.predict()` の戻り値は、10個の数値です。
これは、インデックスに対応するクラスである確率を表しています。
0番目の要素がクラス0（airplane：飛行機）である確率、
1番目の要素がクラス1（automobile：自動車）である確率 … という具合です。

推論結果を見てみましょう。

```
np.set_printoptions(suppress=True)  # ndarrayをprint()した時に指数表記しない

# 最初のテストデータに対する推論結果
print(y_pred[0])
```

```
[0.00725065 0.16515839 0.01526058 0.3431847  0.00276589 0.26259407
 0.07100509 0.00309915 0.04540357 0.08427792]
```

今回は一番確率が高かったクラスに分類することにします。
numpy.ndarray の中で一番大きい値のインデックスを取得するには、`np.argmax()` を使用します。

```
# 最も確率が高いクラスの値
pred_class = np.argmax(y_pred[0])

print('最も確率が高いクラスの値: ', pred_class)
print('クラス名: ', class_names[pred_class])
```

```
最も確率が高いクラスの値:  3
クラス名:  cat
```

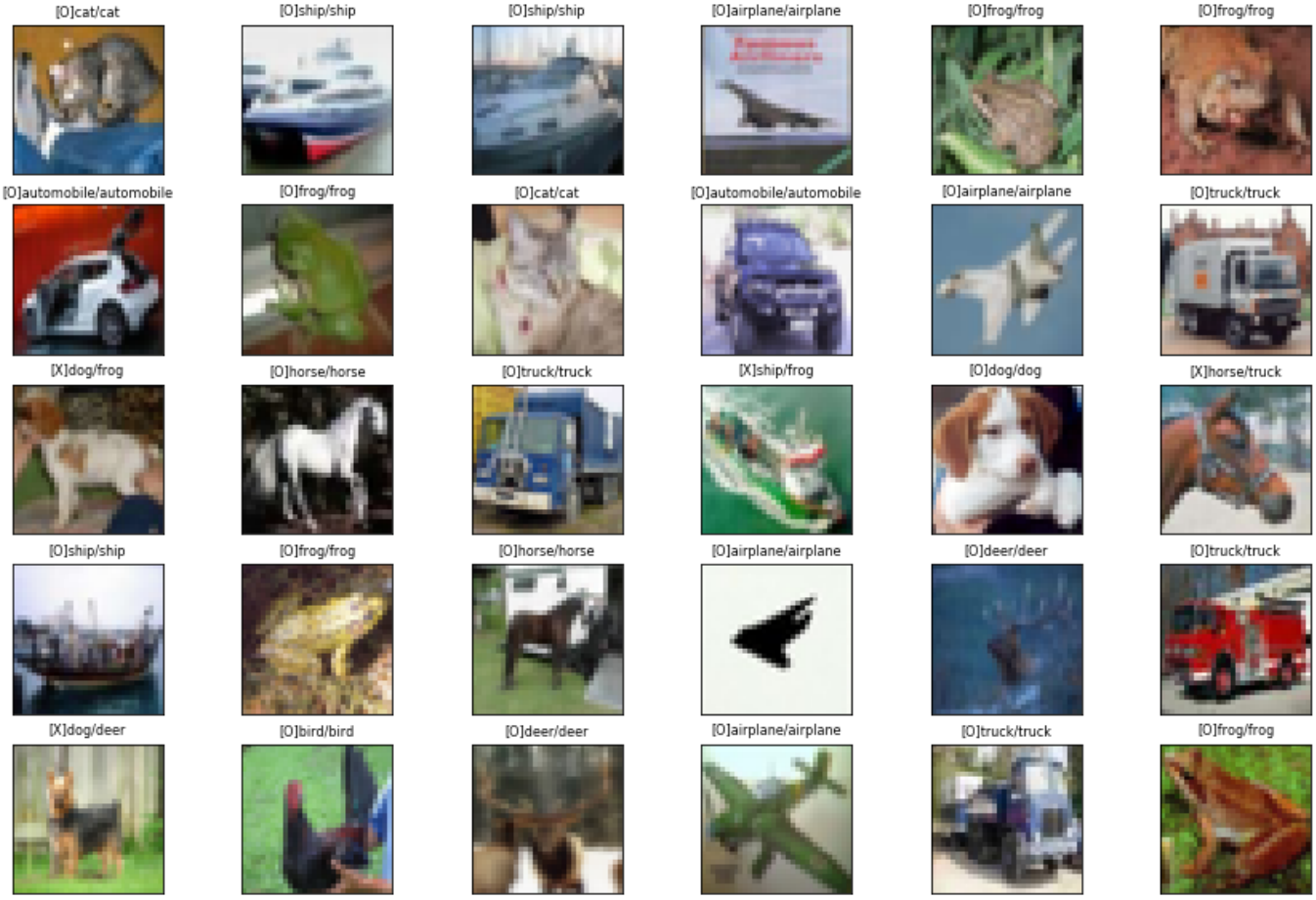
他のテストデータはどうだったか、見てみましょう。

```
# 正解と予測を表示する関数
def show_img_pred(x, y_true, y_p):
  plt.clf()
  plt.figure(figsize=(15, 10))
  for i in range(0, 30):

    title = '[O]' if y_true[i] == y_p[i] else '[X]'
    title += class_names[y_true[i]] + '/' + class_names[y_p[i]]

    plt.subplot(5, 6, i+1)
    plt.title(title, fontsize=8)
    fig = plt.imshow(x[i])
    fig.axes.get_xaxis().set_visible(False)
    fig.axes.get_yaxis().set_visible(False)
```

```
# 確率が高いクラスの値をリストにする
y_pred_flat = [np.argmax(yi) for yi in y_pred]
```

```
# 正解と予測を表示
show_img_pred(x_test, y_test_flat, y_pred_flat)
```

```
<Figure size 432x288 with 0 Axes>
```



評価

`sklearn.metrics.classification_report()` を使って、分類の精度を評価しましょう。

```
# 正解と予測値のクラス名リストを作る
y_test_class_names = [class_names[x] for x in y_test_flat]  # 正解のラベル
y_pred_class_names = [class_names[x] for x in y_pred_flat]  # 予測結果

# 指標の表示
print(classification_report(y_test_class_names, y_pred_class_names))
```

```
              precision    recall  f1-score   support

    airplane       0.73      0.70      0.72      1000
  automobile       0.69      0.83      0.75      1000
        bird       0.70      0.48      0.57      1000
         cat       0.59      0.34      0.43      1000
        deer       0.65      0.52      0.58      1000
         dog       0.71      0.45      0.55      1000
        frog       0.50      0.90      0.64      1000
       horse       0.70      0.77      0.73      1000
        ship       0.85      0.73      0.79      1000
       truck       0.61      0.83      0.70      1000
```

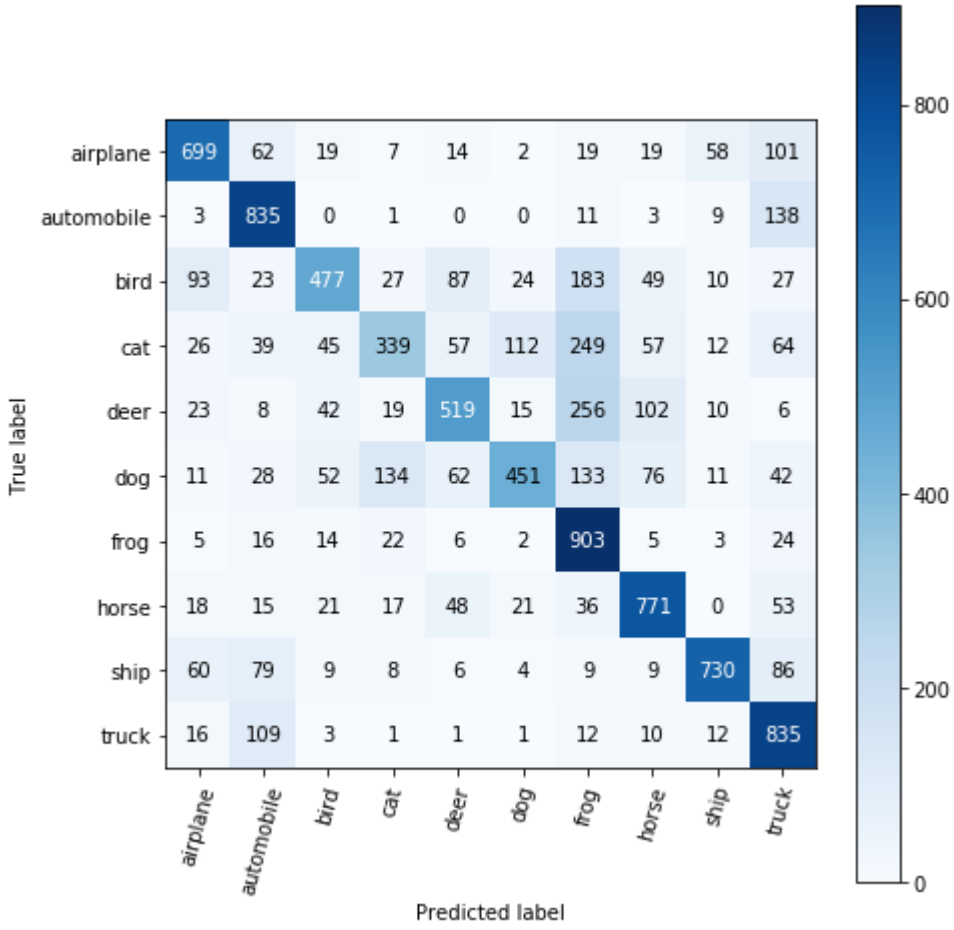
```
    accuracy                           0.66     10000
   macro avg       0.67      0.66      0.65     10000
weighted avg       0.67      0.66      0.65     10000
```

それぞれのクラスが何に誤分類されているのか、混同行列で見てみましょう。

```
# 混同行列を描画する関数
def plot_confusion_matrix(test_list, pred_list):

  # 混同行列を作成
  cm = confusion_matrix(test_list, pred_list)

  fig, ax = plt.subplots(figsize=(7,7))
  im = ax.imshow(cm, interpolation='nearest', cmap='Blues')
  ax.figure.colorbar(im, ax=ax)
  ax.set(xticks=np.arange(cm.shape[1]),
         yticks=np.arange(cm.shape[0]),
         xticklabels=class_names.values(), yticklabels=class_names.values(),  #
クラス名を表示
         ylabel='True label',
         xlabel='Predicted label')

  thresh = cm.max() / 2.
  for i in range(cm.shape[0]):
      for j in range(cm.shape[1]):
          ax.text(j, i, cm[i, j],
                  ha="center", va="center",
                  color="white" if cm[i, j] > thresh else "black")

  # x軸のラベルを回転
  for ax in fig.axes:
      plt.sca(ax)
      plt.xticks(rotation=75)

  fig.tight_layout()

  return ax
```

```
# 混同行列を描画
plot_confusion_matrix(y_test_class_names, y_pred_class_names)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f88cca27438>
```



チューニング

先ほどのモデルは良い精度とは言えませんでした。
学習の推移をみると、訓練データに対しても損失が十分に収束していませんので、学習が足りていないことがわかります。
単純にエポック数を増やしてみましたが、途中から学習が進まなくなりました。
モデルが小さすぎてモデルの表現力が足りていないと考えられますので、もう少し深いニューラルネットワークにしてみます。
一方、モデルを大きくすると過学習が発生しやすくなります。
そこで、隠れ層では途中にバッチ正規化を行う `BatchNormalization` 層を挟みました。
先程はDense層に渡す入力を１次元にするために、多次元の入力を単純に1次元に変換する `Flatten` 層を用いましたが、変換の過程でも特徴を抽出できるよう、平均プーリングを行って１次元の出力を行う `GlobalAveragePooling2D` 層に置き換えてみました。
正規化やドロップアウトを行っている分、学習がゆっくり進みますので、エポック数を増やしました。
（実行にはGoogleColaboratoryで2時間以上かかりました）

```
# # モデルの定義
# model = tf.keras.Sequential([

#     # --- 隠れ層 --- #

#     layers.Conv2D(32, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu', input_shape=(32,32,3)),
#     layers.Conv2D(32, kernel_size=(3,3), strides=(1,1), padding='same',
```

```
activation='relu'),
#     layers.BatchNormalization(),
#     layers.Conv2D(32, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu'),
#     layers.MaxPool2D(pool_size=(2,2), strides=(2,2)),

#     layers.Conv2D(64, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu'),
#     layers.Conv2D(64, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu'),
#     layers.BatchNormalization(),
#     layers.Conv2D(64, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu'),
#     layers.MaxPool2D(pool_size=(2,2), strides=(2,2)),

#     layers.Conv2D(128, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu'),
#     layers.Conv2D(128, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu'),
#     layers.BatchNormalization(),
#     layers.Conv2D(128, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu'),

#     layers.GlobalAveragePooling2D(),

#     # --- 出力層 --- #

#     # 結合層
#     layers.Dense(1024, activation='relu'),
#     layers.Dropout(0.4),
#     layers.Dense(1024, activation='relu'),
#     layers.Dropout(0.4),
#     layers.Dense(128, activation='relu'),
#     layers.Dropout(0.2),
#     layers.Dense(num_classes, activation='softmax'),

# ])

# # モデルのコンパイル
# model.compile(optimizer='adam',
#               loss='categorical_crossentropy',
#               metrics=['accuracy'])

# # モデルの構造を出力
# model.summary()
```

```
# # チェックポイント
# checkpoint = ModelCheckpoint('/content/drive/My Drive/AI_lessons/2-
5_2_classification_cifar10/model/cifar10_improve_model.h5',
#                              monitor='val_accuracy',
#                              save_best_only=True)  # val_accuracyが最も良かったモ
```

```
デルを保存

# # 学習とエポックごとの評価
# history = model.fit_generator(train_generator,      # 訓練データの
ImageDataGenerator
#                     epochs=200,                     # エポック数
#                     validation_data=(x_val,y_val),  # 検証データ
#                     callbacks=[checkpoint])         # エポックごとにチェックポイン
ト保存
```

上記で作成したモデルを読み込んで、評価してみます。

```
# モデルの読み込み
# cifar10_improve_model.h5のパス
model = models.load_model('/content/drive/My Drive/AI_lessons/2-
5_2_classification_cifar10/cifar10_improve_model.h5')

# モデルの構造
model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_87 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_88 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_27 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_89 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_90 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_91 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_28 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_92 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_93 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_94 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
```

```
batch_normalization_29 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_95 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
global_average_pooling2d_2 ( (None, 128)               0
_________________________________________________________________
dense_38 (Dense)             (None, 1024)              132096
_________________________________________________________________
dropout_23 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_39 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dropout_24 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_40 (Dense)             (None, 128)               131200
_________________________________________________________________
dropout_25 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_41 (Dense)             (None, 10)                1290
=================================================================
Total params: 1,795,850
Trainable params: 1,795,402
Non-trainable params: 448
_________________________________________________________________
```

```
# 推論
y_pred = model.predict(x_test)
```

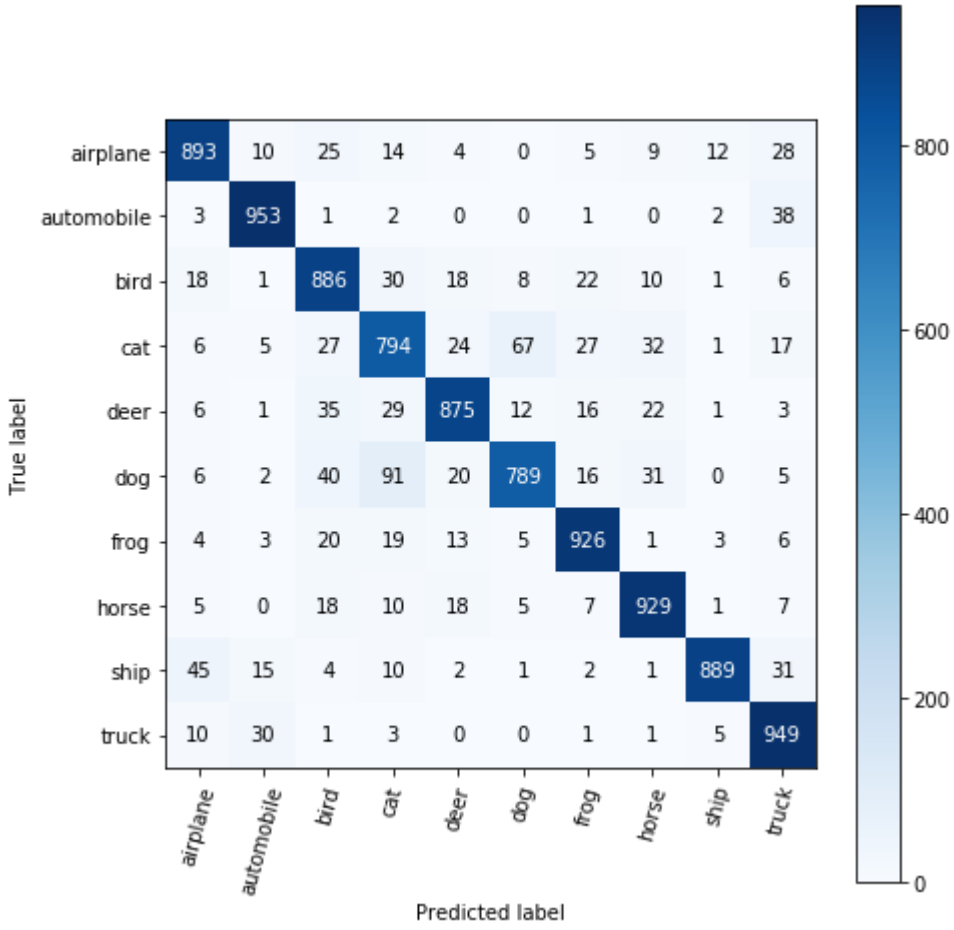
```
# 予測値のクラス名リストを作る
y_pred_class_names = [class_names[np.argmax(x)] for x in y_pred]  # 予測結果

# 指標の表示
print(classification_report(y_test_class_names, y_pred_class_names))
```

```
              precision    recall  f1-score   support

    airplane       0.90      0.89      0.89      1000
  automobile       0.93      0.95      0.94      1000
        bird       0.84      0.89      0.86      1000
         cat       0.79      0.79      0.79      1000
        deer       0.90      0.88      0.89      1000
         dog       0.89      0.79      0.84      1000
        frog       0.91      0.93      0.92      1000
       horse       0.90      0.93      0.91      1000
        ship       0.97      0.89      0.93      1000
```

```
           truck       0.87      0.95      0.91      1000

        accuracy                           0.89     10000
       macro avg       0.89      0.89      0.89     10000
    weighted avg       0.89      0.89      0.89     10000
```

```
# 混同行列を描画
plot_confusion_matrix(y_test_class_names, y_pred_class_names)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f88565f84e0>
```



モデルの構造を変えたことで、正解率は89%まで向上しました。

他にも、学習がある程度進んだら学習率を小さくする「学習率減衰」や、複数のモデルを組み合わせて多数決をとる「アンサンブル学習」などの手法もあります。

オリジナルのデータセットを作る

ImageDataGeneratorを使って、オリジナルのデータセットを作ることができます。
フォルダに以下の構造で画像を格納し、`flow_from_directory()` メソッドを使って読み込みます。

画像を入れるディレクトリ/
│
├ クラス名1/
│ 画像
│ 画像
│ 画像
│
├ クラス名2/
│ 画像
│ 画像
│ 画像
…

作成したデータセットを使って学習したり、推論することができます。
cifar10と同じクラスの画像を集めましたので、推論に使ってみましょう。

```
# my_images.zipのパス ※My Driveのスペースの前に「\」をつける
data_path = '/content/drive/My\ Drive/AI_lessons/2-
5_2_classification_cifar10/my_images.zip'

# 作業ディレクトリを作成してzipを展開
!mkdir /content/cifar10_my_imgaes
!unzip -d /content/cifar10_my_imgaes {data_path}
```

```
Archive:  /content/drive/My Drive/AI_lessons/2-
5_2_classification_cifar10/my_images.zip
  inflating: /content/cifar10_my_imgaes/airplane/airplane1.jpg
  inflating: /content/cifar10_my_imgaes/airplane/airplane2.jpg
  inflating: /content/cifar10_my_imgaes/airplane/airplane3.jpg
  inflating: /content/cifar10_my_imgaes/automobile/automobile1.jpg
  inflating: /content/cifar10_my_imgaes/automobile/automobile2.jpg
  inflating: /content/cifar10_my_imgaes/automobile/automobile3.jpg
  inflating: /content/cifar10_my_imgaes/bird/bird1.jpg
  inflating: /content/cifar10_my_imgaes/bird/bird2.jpg
  inflating: /content/cifar10_my_imgaes/bird/bird3.jpg
  inflating: /content/cifar10_my_imgaes/cat/cat1.jpg
  inflating: /content/cifar10_my_imgaes/cat/cat2.jpg
  inflating: /content/cifar10_my_imgaes/cat/cat3.jpg
  inflating: /content/cifar10_my_imgaes/deer/deer1.jpg
  inflating: /content/cifar10_my_imgaes/deer/deer2.jpg
  inflating: /content/cifar10_my_imgaes/deer/deer3.jpg
  inflating: /content/cifar10_my_imgaes/dog/dog1.jpg
  inflating: /content/cifar10_my_imgaes/dog/dog2.jpg
  inflating: /content/cifar10_my_imgaes/dog/dog3.jpg
  inflating: /content/cifar10_my_imgaes/frog/frog1.jpg
  inflating: /content/cifar10_my_imgaes/frog/frog2.jpg
  inflating: /content/cifar10_my_imgaes/frog/frog3.jpg
```

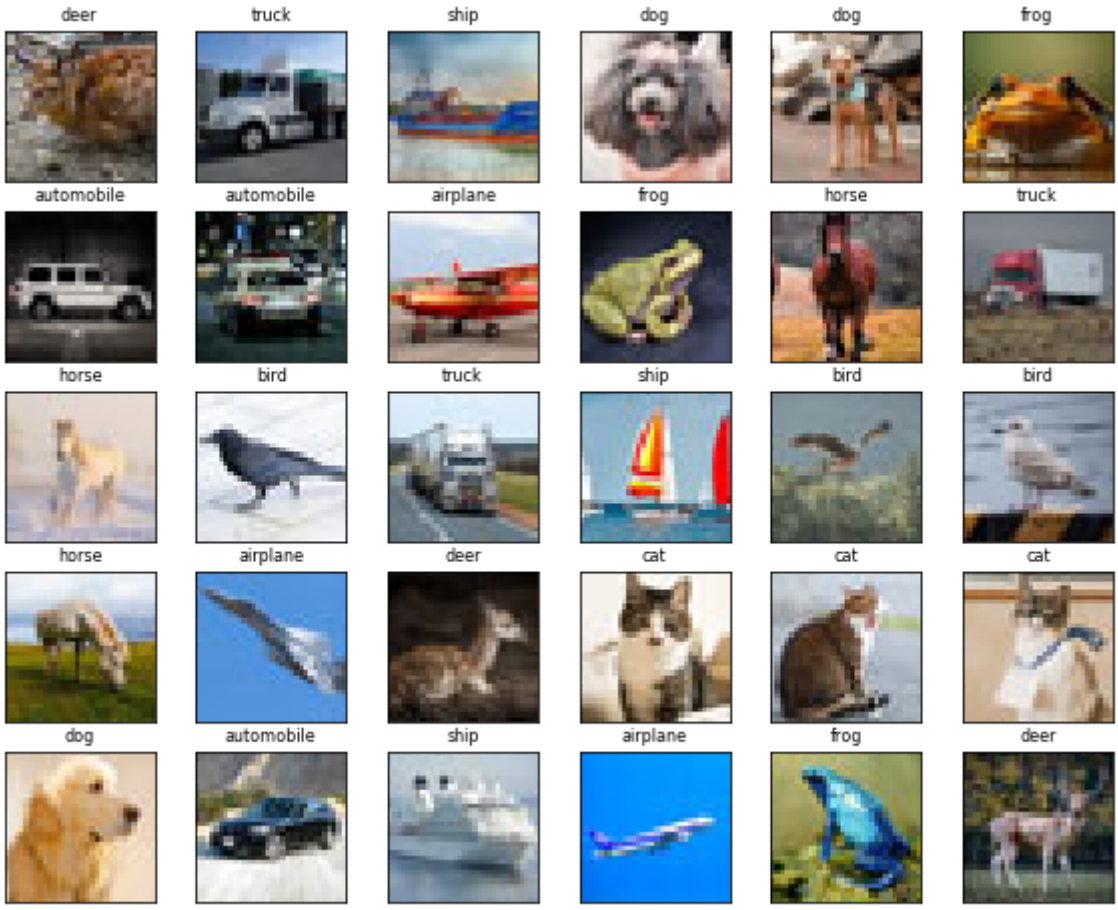
```
  inflating: /content/cifar10_my_imgaes/horse/horse1.jpg
  inflating: /content/cifar10_my_imgaes/horse/horse2.jpg
  inflating: /content/cifar10_my_imgaes/horse/horse3.jpg
  inflating: /content/cifar10_my_imgaes/ship/ship1.jpg
  inflating: /content/cifar10_my_imgaes/ship/ship2.jpg
  inflating: /content/cifar10_my_imgaes/ship/ship3.jpg
  inflating: /content/cifar10_my_imgaes/truck/truck1.jpg
  inflating: /content/cifar10_my_imgaes/truck/truck2.jpg
  inflating: /content/cifar10_my_imgaes/truck/truck3.jpg
```

```
# フォルダから画像を読み込む
datagen = ImageDataGenerator(rescale=1/255.0)  # 0～1の範囲にリスケール
generator = datagen.flow_from_directory('/content/cifar10_my_imgaes',
                                        target_size=(32,32),
                                        color_mode='rgb')
```

```
Found 30 images belonging to 10 classes.
```

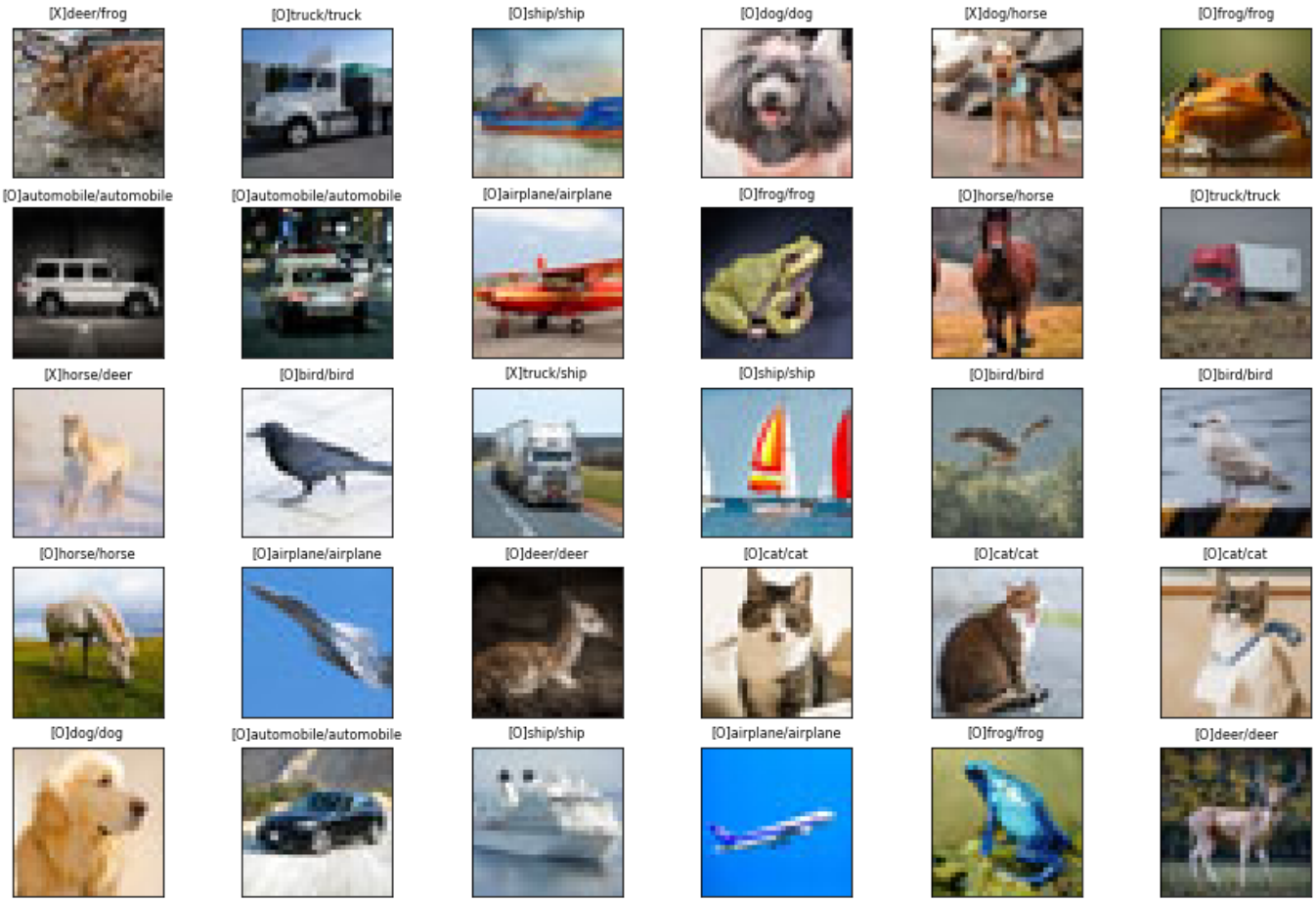
```
# 読み込んだ画像を表示する
x, y = generator.next()
y_flat = [np.argmax(yi) for yi in y]
show_img(x, y_flat)
```

```
<Figure size 432x288 with 0 Axes>
```




```
# ImageDataGeneratorを使って推論
y_pred = model.predict_generator(generator)

# 確率が高いクラスの値をリストにする
y_pred_flat = [np.argmax(yi) for yi in y_pred]

# 正解と予測を表示
show_img_pred(x, y_flat, y_pred_flat)
```

```
<Figure size 432x288 with 0 Axes>
```



オリジナルのデータセットを作って、ぜひ分類に挑戦してみて下さい。

まとめ

ここまで、以下のことを学習しました。

- 画像データの仕組み
  画像は、ピクセルごとの色情報が集まってできています。
  色はRGBの3チャンネルで、それぞれ0～255の値で表されます。

- データ拡張（Data Augmentation）
  画像を回転させたり、引き伸ばしたりすることで、データ数を水増し、汎化性能を向上します。
  ただし、加工によって画像の意味が変わらないように注意します。

- 畳み込みニューラルネットワーク（CNN）の実装方法
  畳み込みニューラルネットワークの隠れ層では、畳み込み層とプーリング層を重ねることで、画像の特徴を抽出していきます。
  モデルの出力は「どのクラスに属するか」を確率で表したいので、出力層はDence層、活性化関数はsoftmaxを使用します。

- 学習中のチェックポイントの作成方法と、学習を途中から再開する方法

  - チェックポイントの保存
    `fit()` メソッドで学習を行うときに `callbacks` で毎エポック終了時に行う処理を指定できます。
    `callbacks` に `ModelCheckpoint` を指定すると、学習途中のモデルを保存できます。

    - 保存した重みを読み込む方法
      `load_weights()` メソッドで保存した重みを読み込むことができます。
      その後、 `fit()` で追加の学習を行います。

- オリジナルのデータセットの作り方
  `ImageDataGenerator` を使って、フォルダに格納されている画像からオリジナルのデータセットを作ることができます。

物体検出（YOLOv3）

「YOLO」を使って、物体検出をしてみましょう。

ここで学習すること

- 既存のモデルを使用した物体検出
- オリジナルのデータを使用した物体検出
- 物体検出の評価指標

YOLO（You Only Look Once）とは

YOLOは広く使われている物体検出アルゴリズムで、Joseph Redmon氏が発表しました。
Redmon氏自身がYOLOをC言語で実装したものがdarknetです。
現在はバージョン 3 が公開されています。

YOLO: Real-Time Object Detection
https://pjreddie.com/darknet/yolo/

GitHub
https://github.com/pjreddie/darknet

2017年にRedmon氏はTEDに出演し、YOLOのプレゼンを行っています。興味のある方は見てみてください。
https://www.drsgate.com/company/c00052/231.php?&ref=rnq

YOLO（darknet）はオープンソースですので、様々な拡張実装が存在します。

- Windows・Linuxで動作可能にしたもの
  GitHub Yolo-v3 and Yolo-v2 for Windows and Linux
  https://github.com/AlexeyAB/darknet
- Keras版
  keras-yolo3
  https://github.com/qqwweee/keras-yolo3
- Tensorflow版
  darkflow　※YOLOv3には未対応
  https://github.com/thtrieu/darkflow

ソースのダウンロードとビルド

今回は AlexeyAB 版のdarknetを使用します。ソースはGitHubから取得します。
GPUの使用など環境に合わせて設定ファイルを書き換え、ビルドします。

```
# ソースをダウンロード
!git clone https://github.com/AlexeyAB/darknet/

%cd darknet

# オプションを指定
!sed -i 's/OPENCV=0/OPENCV=1/g' Makefile    # OpenCVを使用する
!sed -i 's/GPU=0/GPU=1/g' Makefile          # GPUを使用する
!sed -i 's/LIBSO=0/LIBSO=1/g' Makefile      # SOファイルを出力する

# ビルドする
!make


Cloning into 'darknet'...

～ 出力省略 ～
```

darknetで使用するファイル

darknetはモデルの構成や重みをファイルで扱います。
推論を行うには、以下の種類のファイルを用意します。

- dataファイル（.data） … クラスの数や、各種ファイルのパスを記載
- cfgファイル（.cfg） … モデルの構造や、各種パラメータを記載
- 重みファイル（.weights） … 訓練した重み

darknetには、オープンなデータセットを使って訓練されたモデルが用意されています。

データセット	dataファイル	cfgファイル	重みファイル（ダウンロード）
COCO	cfg/coco.data	cfg/yolov3.cfg	https://pjreddie.com/media/files/yolov3.weights
Open Images dataset	cfg/openimages.data	cfg/yolov3-openimages.cfg	https://pjreddie.com/media/files/yolov3-openimages.weights

訓練済みモデルで物体検出（静止画）

COCOデータセットで訓練されたモデルを使って、静止画から物体検出をしてみましょう。

重みファイルをダウンロードします。

```
# 重みをダウンロード
!wget https://pjreddie.com/media/files/yolov3.weights
```

```
--2019-11-25 02:53:38--  https://pjreddie.com/media/files/yolov3.weights
Resolving pjreddie.com (pjreddie.com)... 128.208.4.108
Connecting to pjreddie.com (pjreddie.com)|128.208.4.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 248007048 (237M) [application/octet-stream]
Saving to: ‘yolov3.weights’

yolov3.weights      100%[===================>] 236.52M  23.6MB/s    in 11s

2019-11-25 02:53:50 (21.8 MB/s) - ‘yolov3.weights’ saved [248007048/248007048]
```

サンプル画像を使って、物体検出を試してみましょう。

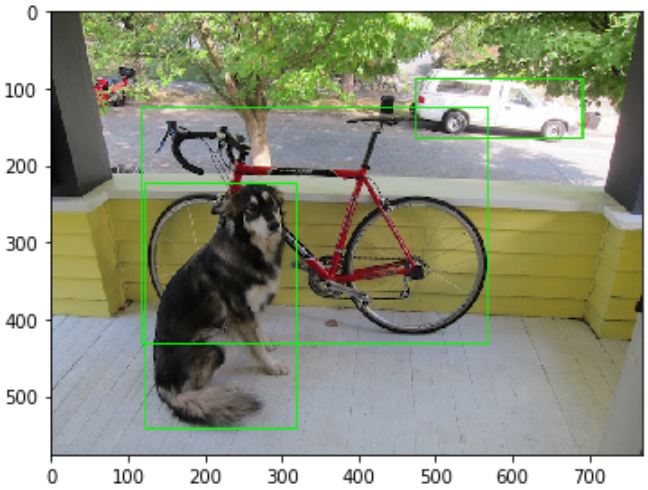
```
from IPython.display import Image, display

# サンプル画像を表示
display(Image('data/dog.jpg'))
```

```
# pythonで実行するための準備

# ライブラリのインストール
!pip install scikit-image

# darknetをimport
!export PYTHONPATH='/content/darknet'
import darknet
```

```
Requirement already satisfied: scikit-image in /usr/local/lib/python3.6/dist-packages (0.15.0)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.0 in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image
Requirement already satisfied: PyWavelets>=0.4.0 in /usr/local/lib/python3.6/dist-packages (from scikit-i
Requirement already satisfied: networkx>=2.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image
Requirement already satisfied: imageio>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from scikit-imag
Requirement already satisfied: pillow>=4.3.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib!=3
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.6/dist-packages (from matplotlib!=3.
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matpl
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotl
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.6/dist-packages (from networkx>
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist-packages (from pillow>=4.3.0->sci
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from cycler>=0.10->matplotl
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from kiwisolver>=1.0
```

```
# dataファイル
meta_path = './cfg/coco.data'
# cfgファイル
config_path = './cfg/yolov3.cfg'
# 重みファイル
weight_path = 'yolov3.weights'

# 初期化
darknet.performDetect(initOnly=True,              # 初期化のみ行う
                      metaPath=meta_path,         # dataファイル
                      configPath=config_path,     # cfgファイル
                      weightPath=weight_path)     # 重みファイル


def detect_image(image_path, meta_path, config_path, weight_path):
  '''静止画の物体検出

  Args:
    image_path (str): 画像ファイルのパス
  '''

  # 物体検出
  d = darknet.performDetect(imagePath=image_path,          # 検出にかける画像
                        metaPath=meta_path,                # dataファイル
                        configPath=config_path,            # cfgファイル
                        weightPath=weight_path)            # 重みファイル

  # 結果を表示
  for detection in d['detections']:
    print('ラベル：', detection[0], end=' ')
    print('信頼度（confidence）：', detection[1], end=' ')
    print('座標（X, Y, 幅, 高さ）：', detection[2])


Initialized detector


# 静止画の物体検出
detect_image('data/dog.jpg', meta_path, config_path, weight_path)
```

```
*** 3 Results, color coded by confidence ***
dog: 100.0%
bicycle: 99.0%
truck: 94.0%
```




```
ラベル： dog 信頼度（confidence） ： 0.9978053569793701 座標（X, Y, 幅, 高さ） ： (221.8568115234375, 383.3585
ラベル： bicycle 信頼度（confidence） ： 0.98981773853302 座標（X, Y, 幅, 高さ） ： (343.3912048339844, 278.48
ラベル： truck 信頼度（confidence） ： 0.9374105334281921 座標（X, Y, 幅, 高さ） ： (582.4186401367188, 126.84
```

darknetは実行ファイルですので、コマンドから実行することもできます。

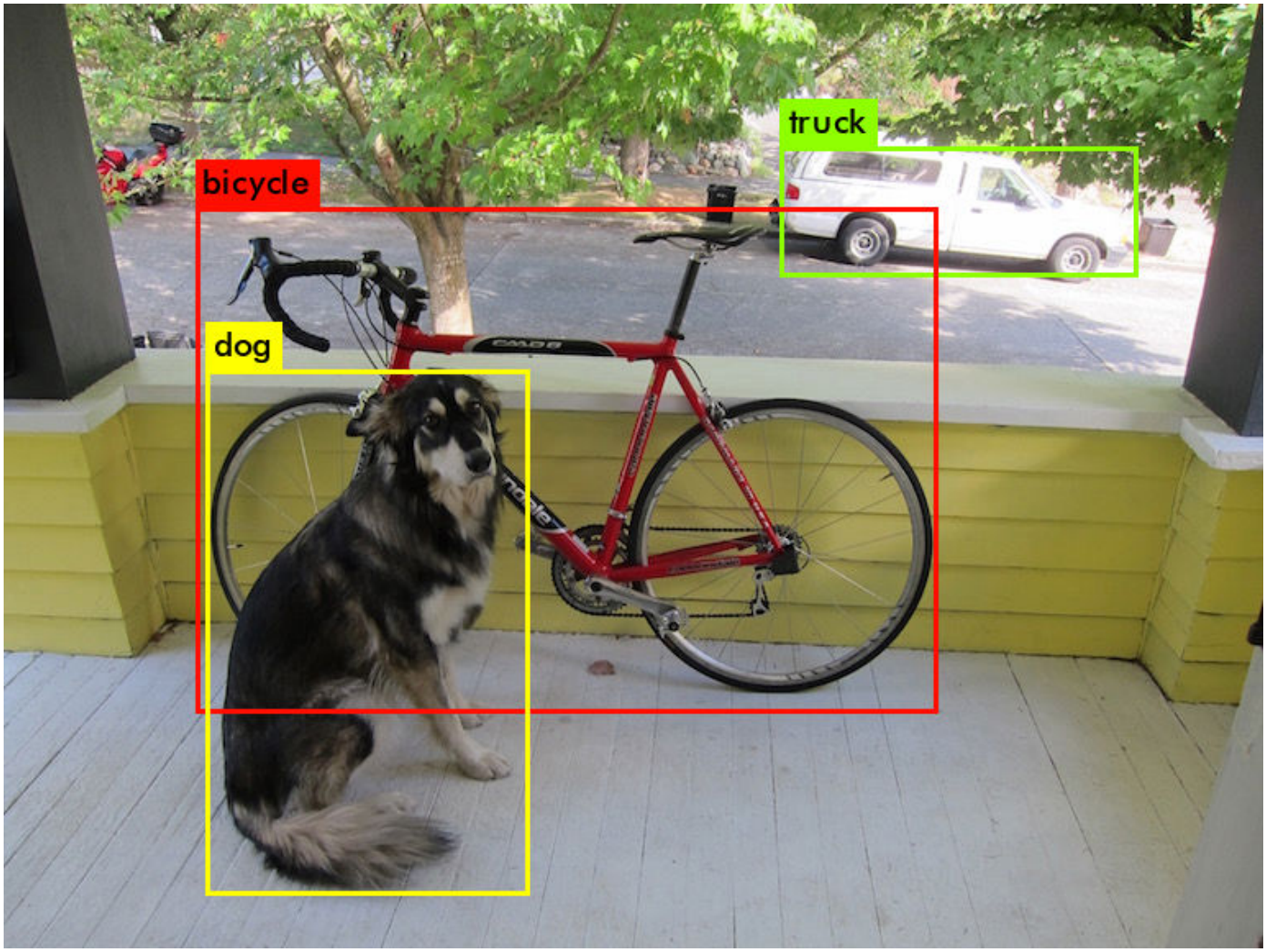
```
!./darknet detector test dataファイル cfgファイル 重みファイル（.weights） オプション 画像ファイル名
```

```
# 画像の物体検出
!./darknet detector test cfg/coco.data cfg/yolov3.cfg yolov3.weights -ext_output data/dog.jpg
```

```
batch = 1, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF

   ～ 出力省略 ～

data/dog.jpg: Predicted in 263.280000 milli-seconds.
bicycle: 99%    (left_x:  117   top_y:  124   width:  452   height:  309)
dog: 100%       (left_x:  124   top_y:  224   width:  196   height:  320)
truck: 94%      (left_x:  474   top_y:   87   width:  217   height:   79)
Unable to init server: Could not connect: Connection refused

(predictions:1126): Gtk-□[1;33mWARNING□[0m **: □[34m02:54:41.255□[0m: cannot open display:
```

検出結果のバウンディングボックスを描き込んだ画像が predictions.jpg というファイル名で保存されています。
predictions.jpg を表示してみましょう。

```
# 検出結果の画像を表示
display(Image('predictions.jpg'))
```



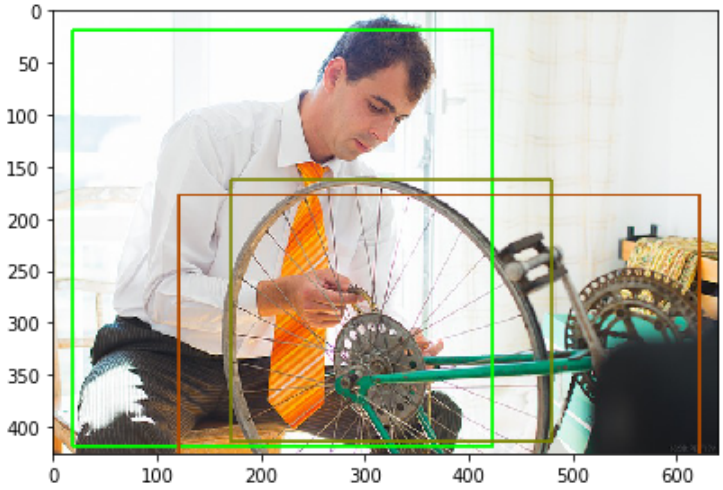
好きな画像で物体検出

好きな画像ファイルをアップロードして、物体検出してみましょう。

```
# ファイルアップロード
from google.colab import files
files = files.upload()
uploaded_file_name = list(files.keys())[0]

# 静止画の物体検出
detect_image(uploaded_file_name, meta_path, config_path, weight_path)
```

```
 <input type="file" id="files-5244b105-77ae-4ea7-b679-4f87633fe97a" name="files[]" multiple disabled />
 <output id="result-5244b105-77ae-4ea7-b679-4f87633fe97a">
  Upload widget is only available when the cell has been executed in the
  current browser session. Please rerun this cell to enable.
  </output>
  <script src="/nbextensions/google.colab/files.js"></script>
```

```
Saving man-1148982_640.jpg to man-1148982_640.jpg
*** 3 Results, color coded by confidence ***
person: 100.0%
tennis racket: 72.0%
bicycle: 54.0%
```




```
ラベル： person 信頼度（confidence） ： 0.9993019700050354 座標（X, Y, 幅, 高さ） ： (220.95278930664062, 219.
ラベル： tennis racket 信頼度（confidence） ： 0.7194201946258545 座標（X, Y, 幅, 高さ） ： (326.2222290039062
ラベル： bicycle 信頼度（confidence） ： 0.5449868440628052 座標（X, Y, 幅, 高さ） ： (372.2989501953125, 302.
```

訓練済みモデルで物体検出（動画）

動画から物体検出する機能も提供されています。

```
# Googleドライブをマウント
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf

Enter your authorization code:
··········
Mounted at /content/drive
```

```
# Googleドライブから動画をコピー
# video.mp4のパス ※My Diveのスペースの前に「\」を入れる
VIDEO_PATH = '/content/drive/My\ Drive/AI_lessons/2-5_3_recognition/video.mp4'

!cp {VIDEO_PATH} .
```

```
# 動画の物体検出
!./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights -dont_show video.mp4 -i 0 -out_filen
```

```
Demo
batch = 1, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF

～ 出力省略 ～

video file: video.mp4
Video stream: 3840 x 2160
Objects:

FPS:0.0
Objects:

pottedplant: 54%
pottedplant: 41%
handbag: 31%
bench: 39%
bus: 99%
person: 99%
person: 92%
person: 92%
person: 85%
person: 78%
person: 27%

FPS:-0.0

 cvWriteFrame

～ 出力省略 ～

input video stream closed.
 closing... closed!output_video_writer closed.


# 出力された動画をGoogleドライブにコピー ※うまく行かない場合は少し時間をおいてから実行
!cp video_out.avi /content/drive/My\ Drive/
```

オリジナルデータの物体検出 - きのこ・たけのこ -

いよいよ、オリジナルの画像を使って物体検出をしてみましょう。
今回の題材は「きのこの山・たけのこ里の検出」です。（※きのこの山、たけのこの里は株式会社明治の商品です）

作成するファイル

ローカルPCで以下のファイルを作成します。

- クラス名一覧ファイル（obj.names）
- 画像ファイル（001.jpg ～ 100.jpg）
- アノテーションファイル（001.txt ～ 100.txt）
- 訓練用の画像一覧ファイル（train.txt）
- テスト用の画像一覧ファイル（test.txt）

- dataファイル（obj.data）
- cfgファイル（yolo-obj.cfg）

※「obj」はチュートリアルに倣ってつけた名前ですので、自由につけて構いません。
https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects

画像データの収集

まずは画像データを用意します。
darknet は JPEG 形式（.jpg）の画像しか扱えませんので、他の形式だった場合は JPEG に変換しておきます。

きのこ・たけのこを5個ずつ写した画像を20枚撮影し、各100サンプル集めました。

データの水増し（Data Augumentation）

用意した画像が少ないため、データの水増しを行います。

元の画像	反転	回転	ランダムな切り抜き
			
- 何も変更されていない画像	- 画像の意味が変わらない軸における反転	- わずかな角度の回転 - 不正確な水平線の校正（calibration）をシミュレートする	- 画像の一部へのランダムなフォーカス - 連続して数回のランダムな切り抜きが可能

カラーシフト	ノイズの付加	情報損失	コントラストの修正
			
- RGBのわずかな修正 - 照らされ方によるノイズを捉える	- ノイズの付加 - 入力画像の品質のばらつきへの耐性の強化	- 画像の一部を無視 - 画像の一部が欠ける可能性を再現する	- 明るさの変化 - 時刻による露出の違いをコントロールする

画像引用：https://github.com/afshinea/stanford-cs-230-deep-learning/blob/master/ja/cheatsheet-deep-learning-tips-tricks.pdf

今回は色味の違う画像で水増しすることにします。
PCA Color Augmentation（主成分分析による色の拡張）により、元の画像のコントラストを活かしたまま、色味・明るさの違う画像を生成します。
使用したノートブックは「pca_color_augumentation.ipynb」です。

データ拡張のついでに、画像サイズも長辺が600px程度になるよう縮小しました。
画像サイズが大きすぎると学習時にメモリ不足でエラーになりますので、300px～600px程度に縮小しておきます。

今回は1枚の画像から色味の異なる5枚の画像を作成し、合計100枚に水増ししました。
これできのこ・たけのこは各500サンプルになりました。
画像ファイルの名前は001.jpg～100.jpgの連番としました。

アノテーション付加

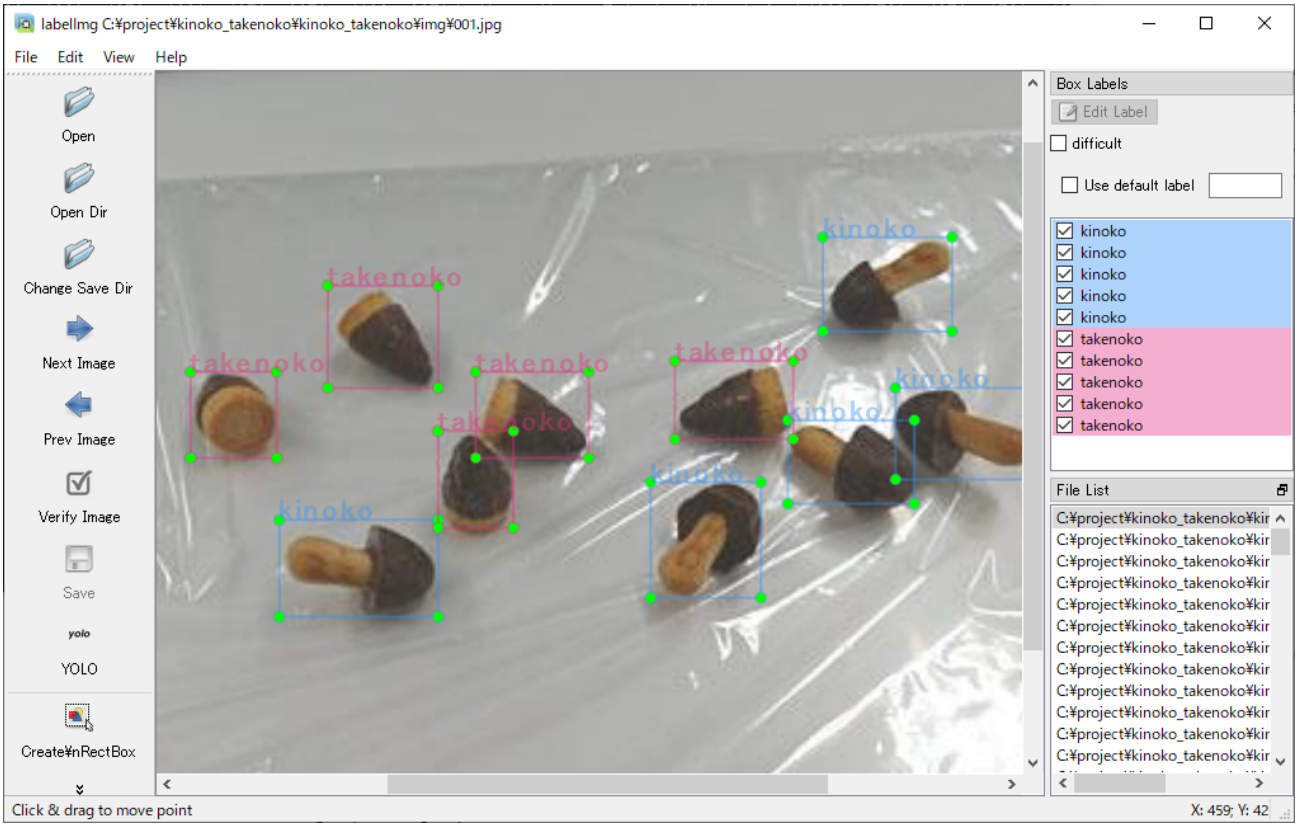
画像に写っている、学習対象の物体の位置を示すデータ「アノテーション」を作成します。
アノテーションの形式は手法によって異なります。
SSDなどXML形式を採用している手法が多いですが、YOLOは独自形式のアノテーションを採用しています。

YOLOのアノテーション形式

```
<object-class> <x_center> <y_center> <width> <height>
```

アノテーションファイルはテキストファイル（.txt）です。
1画像につき1ファイル作成し、ファイル名を一致させます。
例えば、001.jpgに対応するアノテーションファイルは001.txtです。

アノテーションツールも様々です。
今回はYOLO形式のアノテーションが作成できる「LabelImg」というツールを使用しました。
https://github.com/tzutalin/labelImg



他にはMicrosoftのVottが有名です。
※YOLO形式に対応しているのはバージョン1まで。最新のバージョン2系はYOLO形式に対応していないため、コンバートが必要です。
https://github.com/microsoft/VoTT/releases/tag/v1.7.2

LabelImgでアノテーションを作成すると、クラス名の一覧ファイル（classes.txt）と、アノテーション情報が書き込まれた 001.txt ～ 100.txt が出力されます。
classes.txt は darknet 風に「obj.names」にリネームしておきます。

obj.names

```
kinoko
takenoko
```

001.txt ～ 100.txt

```
0 0.379167 0.621302 0.098333 0.106509
0 0.594167 0.590237 0.068333 0.127219
…
1 0.301667 0.452663 0.053333 0.094675
1 0.394167 0.366864 0.068333 0.112426
…
```

画像一覧ファイルの作成

訓練データの画像一覧ファイル（train.txt）と、テストデータの画像一覧ファイル（test.txt）を作成します。
それぞれ、訓練に使う画像ファイルのパスと、テストに使う画像ファイルのパスを列挙します。
パスは darknet（実行ファイル）からの相対パスで指定します。

今回は全部で100枚の画像がありますので、訓練データ80枚、テストデータ20枚としました。

train.txt

```
data/obj/001.jpg
data/obj/002.jpg
data/obj/003.jpg
…
```

test.txt

```
data/obj/005.jpg
data/obj/010.jpg
…
```

dataファイルの作成

dataファイルを作成します。

obj.data

```
classes= 2
train  = data/train.txt
valid  = data/test.txt
names = data/obj.names
backup = backup/
```

- classes … クラス数。きのこ・たけのこの2クラス
- train … train.txt のパス
- valid … test.txt のパス
- names … obj.names のパス
- backup … 学習中の重みファイルのバックアップ保存先パス

パスは darknet（実行ファイル）からの相対パスで指定します。

cfgファイルの作成

モデルの構造の設定ファルを作成します。
１から作成するのは大変なので、以下のURLから既存の設定ファイル「yolov3.cfg」をコピーして、「yolo-obj.cfg」を作成します。
https://github.com/AlexeyAB/darknet/blob/master/cfg/yolov3.cfg

以下の箇所を編集します。

3行目

```
batch=64
```

バッチサイズです。

4行目

```
subdivisions=16
```

通常は8を指定します。メモリが足りずエラーになる場合は、16、32、64のいずれかを設定します。
今回は8だとGoogle Colaboratoryでエラーになったため、16にしています。

20行目

```
max_batches = 4000
```

学習を繰り返す（イテレーション）回数です。
クラス数×2000を設定します。
今回は(2クラス×2000) = 4000です。
※最低4000～なので、1クラスでも4000を指定します。

22行目

```
steps=3200,3600
```

max_batchesの80%、90%の値を指定します。

610行目、696行目、783行目

```
classes=2
```

クラス数を指定します。

603行目、689行目、776行目

```
filters=21
```

(classes + 5)×3 の値を指定します。
今回は(2クラス + 5)×3 = 21です。

作成したファイルをGoogleドライブに格納

作成したファイルは、扱いやすいように zipにまとめて、Googleドライブに格納しておきましょう。

ファイルの配置

ここからは Google Colaboratory 上で作業します。
作業ディレクトリを作成して、Googleドライブから先程作成したzipファイルをコピーし、展開します。

```
# yolo_obj.zipのパス　※My Driveのスペースの前に「\」を入れる
ZIP_PATH = '/content/drive/My\ Drive/AI_lessons/2-5_3_recognition/yolo_obj.zip'

# 作業ディレクトリを作成してGoogleドライブからzipを展開
!mkdir /content/2-5
!unzip -d /content/2-5 {ZIP_PATH}
```

```
Archive:  /content/drive/My Drive/AI_lessons/2-5_3_recognition/yolo_obj.zip

～ 出力省略 ～
```

展開したファイルを darknet ディレクトリの中に配置します。
配置は obj.data や、train.txt、test.txtで指定したパスと一致させます。
今回は以下のように配置します。

```
darknet
├ yolo-obj.cfg
├ data
│ ├ obj.data
│ ├ obj.names
│ ├ train.txt
│ ├ test.txt
│ └ obj
│      ├ 001.jpg
│      ├ 001.txt
│      ├ 002.jpg
│      ├ 002.txt
│      ├ …
```

```
# オリジナルデータを配置
!cp /content/2-5/cfg/yolo-obj.cfg .
!cp -r /content/2-5/data/* ./data
```

訓練済みの重みをダウンロード

１から訓練を行うには、大変な時間がかかります。
そこで、はじめは訓練済みの重みを利用することで、学習を収束させやすくします。

```
# 訓練済みの重みをダウンロード
!wget https://pjreddie.com/media/files/darknet53.conv.74
```

```
--2019-11-25 04:07:07--  https://pjreddie.com/media/files/darknet53.conv.74
Resolving pjreddie.com (pjreddie.com)... 128.208.4.108
Connecting to pjreddie.com (pjreddie.com)|128.208.4.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 162482580 (155M) [application/octet-stream]
Saving to: ‘darknet53.conv.74’

darknet53.conv.74   100%[===================>] 154.96M  23.1MB/s    in 7.6s

2019-11-25 04:07:15 (20.5 MB/s) - ‘darknet53.conv.74’ saved [162482580/162482580]
```

学習

コマンドで学習を実行します。
学習中はログが出力されますが、Google Colaboratory 上では出力が多くなると途中から表示できなくなってしまうため、ファイルに出力します。

```
!./darknet detector train dataファイル cfgファイル 重みファイル（.weights） オプション > ログ出力先
```

```
# 学習
!./darknet detector train data/obj.data yolo-obj.cfg darknet53.conv.74 -dont_show > log.txt
```

```
   layer   filters  size/strd(dil)      input                output
   0 conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF

～ 出力省略 ～

[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
nms_kind: Using default 'default'
Total BFLOPS 65.297
 Allocate additional workspace_size = 49.84 MB
Loading weights from darknet53.conv.74...Done! Loaded 75 layers from weights-file
^C
```

学習中のログは２種類出力されます。

バッチログ

```
  1: 1988.302002, 1988.302002 avg loss, 0.000000 rate, 11.904463 seconds, 64 images Loaded: 0.235293 seconds
```

最初の数字はイテレーション（繰り返し）の回数です。
cfgファイルに「max_batches = 4000」と指定しましたので、4000回繰り返します。
「avg loss」の直前の数値は損失関数の値です。この値が十分に小さくなれば、学習は完了です。
小さく単純なデータセットの場合「0.05」、大きく複雑なデータセットの場合「3」が目安です。
学習の途中であっても、この値が下がらなくなってきたら過学習になる前に学習を止めます。

Subdivisionログ

```
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000)
Region 94 Avg (IOU: 0.360053, GIOU: 0.234920), Class: 0.516058, Obj: 0.626917, No Obj: 0.582338, .5R: 0.250000, .75R: 0.0000
```

「Region 94 Avg (IOU:」の後ろの数値が、性能の評価指標（IoU）の値です。
1に近いほど性能が良いです。（IoUの説明は後ほど行います）
求める性能に達したら学習を止めます。

重みファイルの保存と学習の再開

学習中、100イテレーションごとにbackupフォルダに重みが保存されます（yolo-obj_last.weights）。
また、1000イテレーションごとに重みが別名で保存されます（yolo-obj_1000.weights）。
Google Colaboratoryは起動後12時間でファイルが削除されてしまいます。
学習結果が消えてしまわないよう、ときどき重みファイルをGoogleドライブにコピーしてバックアップしましょう。
翌日などに学習を再開するときには、Googleドライブからbackupフォルダに重みファイルを書き戻します。

```
# # backupに保存された重みをGoogleドライブにコピー
# !cp backup/yolo-obj_last.weights /content/drive/My\ Drive/
```

```
# # セッションが切れてしまったら、Googleドライブから重みを書き戻す
# !cp /content/drive/My\ Drive/yolo-obj_last.weights backup
```

学習を再開するには、backupに保存された重みを使います。

```
# # 学習を続きから再開
# !./darknet detector train data/obj.data yolo-obj.cfg backup/yolo-obj_last.weights -dont_show > log.txt
```

学習が完了すると、backupフォルダに最終の重みファイル（yolo-obj_final.weights）が書き出されますので、保存します。

```
# # backupから最終の重みをコピー
# !cp backup/yolo-obj_final.weights /content/drive/My\ Drive/
# !cp backup/yolo-obj_final.weights .
```

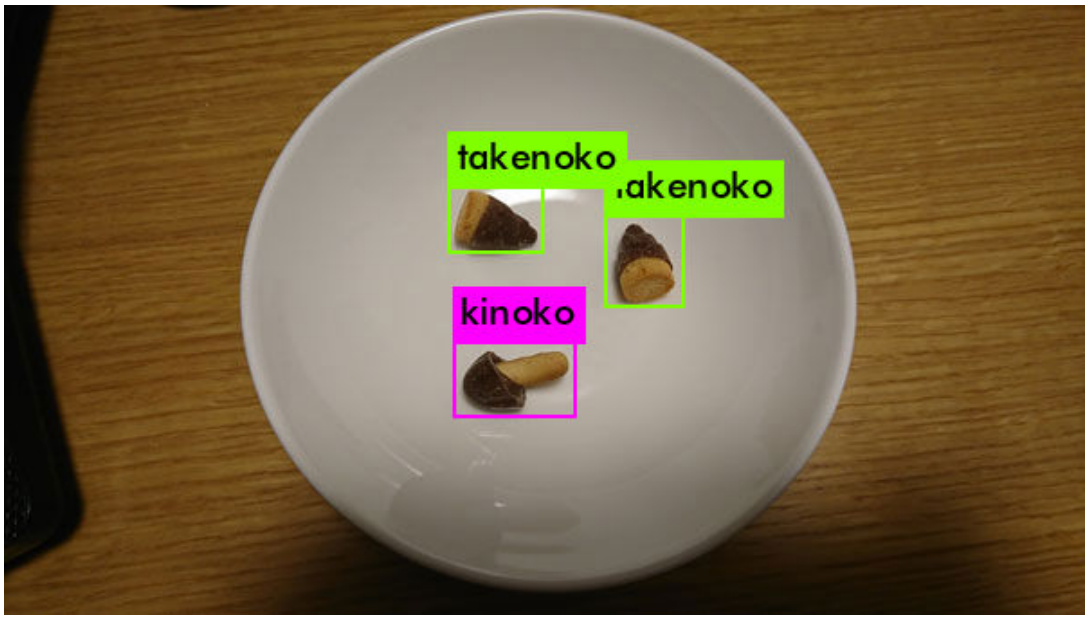
物体検出

作成したモデルを使って、物体検出を試してみましょう。
訓練で使用したdataファイルとcfgファイル、そして最終の重みを使います。

```
# Googleドライブに保存した最終の重みをコピー
# yolo-obj_final.weightのパス　※My Driveのスペースの前に「\」を入れる
FINAL_WEIGHT = '/content/drive/My\ Drive/AI_lessons/2-5_3_recognition/yolo-obj_final.weights'

!cp {FINAL_WEIGHT} .
```

```
# 画像の物体検出
!./darknet detector test data/obj.data yolo-obj.cfg yolo-obj_final.weights -ext_output data/obj/test.jpg

# 検出結果の画像を表示
display(Image('predictions.jpg'))
```

```
batch = 1, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF

～ 出力省略 ～

data/obj/test.jpg: Predicted in 259.489000 milli-seconds.
takenoko: 100%  (left_x:  246   top_y:  100   width:   53   height:   37)
kinoko: 100%    (left_x:  250   top_y:  186   width:   68   height:   42)
takenoko: 100%  (left_x:  334   top_y:  116   width:   43   height:   51)
Unable to init server: Could not connect: Connection refused

(predictions:3958): Gtk-□[1;33mWARNING□[0m **: □[34m04:09:09.291□[0m: cannot open display:
```



評価指標

物体検出は「クラスの検出の正しさ」と「位置の検出の正しさ」を評価する必要があります。
精度の評価指標には以下のようなものがあります。

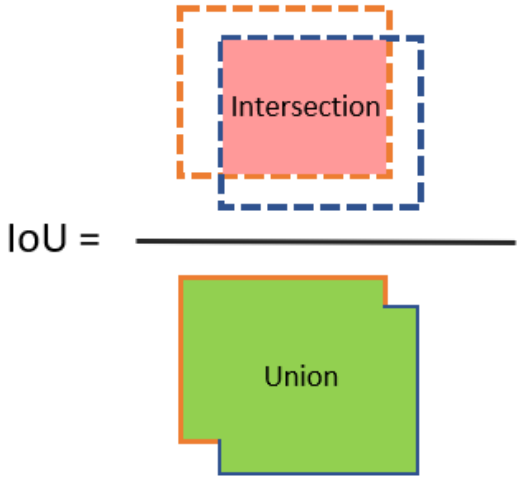
- IoU（Intersection over Union）
- mAP（mean Average Precision）

また、実用ではリアルタイムに推論できることが求められることも多いため、推論速度も重要な評価対象です。
1秒あたりに何枚の画像を処理できるかをFPS（frames per second）で表します。

IoU（Intersection over Union）

IoUは、あるクラスについて、そのクラスの物体であると予測した領域が、どれだけ実際の領域と一致していたかを表す指標です。
値の範囲は0〜1で、1に近いほど正確に予測できたことになります。

$$IoU = \frac{予測が正解した領域(Intersection)}{実際の領域＋予測した領域(Union)}$$



mAP（mean Average Precision）

mAPとはモデルの精度を表す指標です。
0〜1の範囲で、1に近いほど性能が良いです。（%単位で表記することが多いです）
特定のデータセット（PASCAL VOC、COCOなど）で測定され、モデルの性能を比較するのに役立ちます。
例えば以下のような表をよく目にすると思います。

Detector	VOC07 (mAP@IoU=0.5)	VOC12 (mAP@IoU=0.5)	COCO (mAP@IoU=0.5:0.95)	Published In
R-CNN	58.5	-	-	CVPR'14
SPP-Net	59.2	-	-	ECCV'14
MR-CNN	78.2 (07+12)	73.9 (07+12)	-	ICCV'15
Fast R-CNN	70.0 (07+12)	68.4 (07++12)	19.7	ICCV'15
Faster R-CNN	73.2 (07+12)	70.4 (07++12)	21.9	NIPS'15

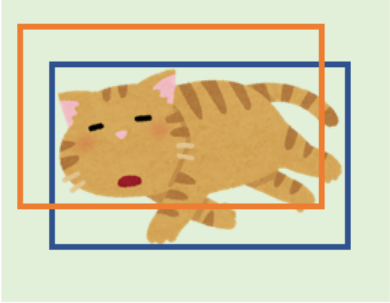
画像引用：https://github.com/hoya012/deep_learning_object_detection

mAPを算出するには、IoU、TP/FP/FN、Precision、APを求める必要があります。
順を追ってみてみましょう。

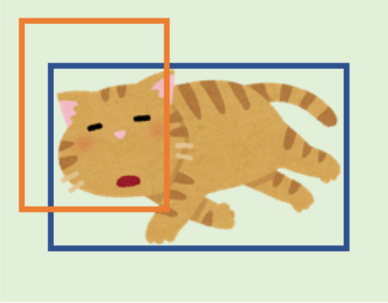
TP（True Positive）、 FP（False Positive）、FN（False Negative）

あるクラスの物体の予測について、IoUを求めます。
そしてIoUの値に一定のしきい値を定め（0.5など）、
しきい値以上であればTP（True Positive）、
しきい値未満であればFP（False Positive）とします。

実際にはそのクラスの物体があるのに検出できなければ、FN（False Negative）とします。



IoU≧0.5 ...TP



IoU＜0.5 ...FP

FN

Precision（適合率）、Recall（再現率）

TP、FP、FNからPrecisionとRecallを算出します。

- Precision … 物体を予測したうち、どれだけ正しく予測できたか

$$Precision = \frac{TP}{TP + FP}$$

- Recall … 実際の物体のうち、どれだけ正しく予測できたか

$$Recall = \frac{TP}{TP + FN}$$

AP（Average Precision）

AP（Average Precision）は、いくつかサンプリングしたPrecisionの平均です。
ただし単純な平均ではなく、誤差が少なくなるように計算されています。
VOCとCOCOでは異なる計算方法を採用しています。詳しくは以下を参照してください。
https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

mAP（mean Average Precision）

APは各クラスについて算出されますので、クラスの平均をとった値がmAPです。
mAPが高くても、クラスによってAPにバラつきがある可能性もありますので、注意が必要です。

IoUのしきい値が変われば、Precisionの値、ひいてはmAPも変わります。
先程の表には「mAP@IoU=0.5」と書かれていました。これは、IoUのしきい値を0.5でPrecisionを算出したという意味です。（「mAP50」と書くこともあります）
「mAP@IoU=0.5」は多少位置がずれていても正解とした精度ですし、「mAP@IoU=0.75」はより位置を正確に検出した精度です。

同じモデルでもIoUのしきい値によって精度が異なる場合があります。
位置の精度まで求める場合は、IoUが高いmAPに注目しましょう。

以下のURLでは、各モデルの論文で公表されているmAPを見ることができます。
https://paperswithcode.com/task/object-detection

まとめ

ここまで、以下のことを学習しました。

- 既存のモデルを使用した物体検出
  トレーニング済みのモデルを使えば、すぐに推論を行うことができます。
- オリジナルのデータを使用した物体検出
  画像を収集し、アノテーションを付加します。
  汎化性能の向上にはデータ拡張（Data Augmentation）が有効です。
- 物体検出の評価指標
  クラスについての指標「IoU」、モデルの評価指標「mAP」があります。
  推論速度（FPS）も重要です。

PCA Color Augumantation

主成分分析を使用した色味のデータ拡張処理

`pca_color_augmentation` 関数のロジックは以下のURLのコードを使用しました。

PCA Color Augmentationを使ってみよう
https://qiita.com/koshian2/items/78de8ccd09dd2998ddfc

```
import numpy as np
from PIL import Image
import glob
```

```
# データ拡張
def pca_color_augmentation(image_array_input):
    assert image_array_input.ndim == 3 and image_array_input.shape[2] == 3
    assert image_array_input.dtype == np.uint8

    img = image_array_input.reshape(-1, 3).astype(np.float32)
    # 分散を計算
    ch_var = np.var(img, axis=0)
    # 分散の合計が3になるようにスケーリング
    scaling_factor = np.sqrt(3.0 / sum(ch_var))
    # 平均で引いてスケーリング
    img = (img - np.mean(img, axis=0)) * scaling_factor

    cov = np.cov(img, rowvar=False)
    lambd_eigen_value, p_eigen_vector = np.linalg.eig(cov)

    rand = np.random.randn(3) * 0.1
    delta = np.dot(p_eigen_vector, rand*lambd_eigen_value)
    delta = (delta * 255.0).astype(np.int32)[np.newaxis, np.newaxis, :]

    img_out = np.clip(image_array_input + delta, 0, 255).astype(np.uint8)
    return img_out
```

```
# Googleドライブをマウント
from google.colab import drive
drive.mount('/content/drive')
```

```
# 画像の格納してあるフォルダに移動
%cd /content/drive/My Drive/images
```

```
RESIZE_HEIGHT = 600   # 画像高さ
RESIZE_WIDTH = 333    # 画像幅
PATTERN = 5           # 1枚の画像から何パターン作成するか

# JPGファイルを検索
images = glob.glob("*.JPG")

count = 0;

for image in images:
  with open(image, 'rb') as file:

    img = Image.open(file)

    # 画像サイズを縮小
    img = img.resize((RESIZE_HEIGHT, RESIZE_WIDTH))

    img_array_input = np.array(img)

    for i in range(PATTERN):
      count += 1

      # データ拡張
      img_array_out = pca_color_augmentation(img_array_input)

      # 拡張した画像を保存
      Image.fromarray(img_array_out).save(f"./pcaca/{count:03}.jpg")
```

```
# ZIPファイルに圧縮
!zip -r pcaca
```

AI2-6

AI の実装（強化学習）

－ 講義内容 －

- 強化学習概論
- 産業応用
- 基礎理論
- 演習： 迷路を解く

強化学習
Reinforcement Learning

2019年12月3日（火）

新潟人工知能研究所 技術開発部

大目次

p03. 科目概要
p06. 概論
p11. 産業応用
p16. 基礎理論

科目概要

株式会社新潟人工知能研究所
技術開発部

シラバス

- 強化学習と教師あり学習・教師なし学習の違いを理解する。
- 強化学習の代表的なアルゴリズムであるQ-Learningの理論を理解して、Pythonによる実装を行う。
- また、その強化学習と深層学習を組み合わせた深層強化学習について、その概要を理解する。
- さらに、「強化学習」の活用事例を通して、自身のビジネスや業務に実装する方法を考える。

強化学習の概要

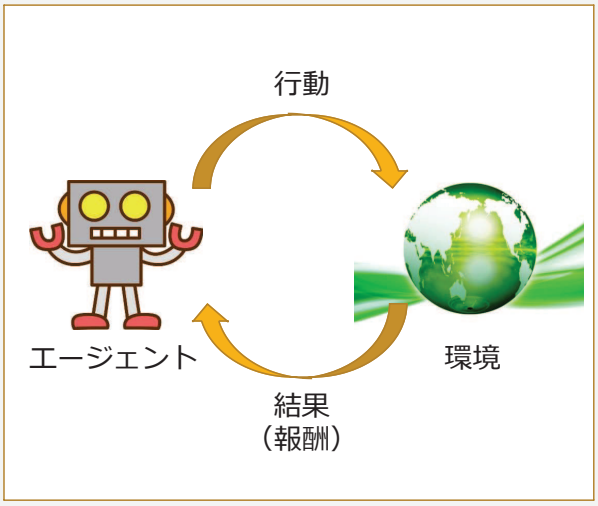
	教師あり学習	教師なし学習	強化学習
学習方法	正解（目的変数）がある訓練データを大量に分析して、パターンを予測するためのモデル（計算式）を導き出す	正解（目的変数）はなく、大量の訓練データの構造を分析し群＝クラスタに分けるモデル（計算式）を導き出す	人工知能の予測に対する誤差をフィードバックすることで、試行錯誤を繰り返しながら精度を自動で高めていく
学習内容	説明変数（入力）と目的変数（出力）の関係	データの構造 データの類似性	報酬（誤差が少ない場合に得られるスコア）を最大化する行動
具体的な手法	回帰モデル 分類モデル	クラスタリング 主成分分析(次元削減)	Q学習 Deep Q-Network
用途の具体例	需要予測 解約予測 迷惑メールフィルタ 画像認識…	顧客特性分析 レコメンドエンジン データ分析の前処理…	株・FXなどのトレード 囲碁・将棋などのボードゲーム 自動運転や工作機械

概論

株式会社新潟人工知能研究所
技術開発部

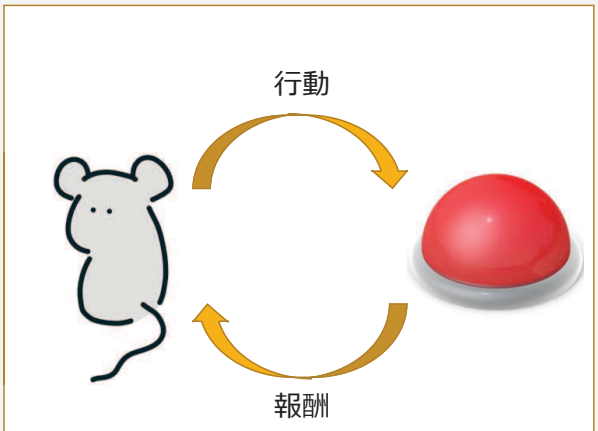
強化学習とは

- 教師あり学習のように正解（教師データ）はズバリ存在しない。
  - かといって、教師なし学習のように正解が存在しないわけでもない。
- 望ましい結果が得られた場合には「報酬」を与えることで学習
  - 歩行ロボットであれば「転ばずに歩けた距離」が、将棋や囲碁の場合は「勝利」が、報酬に相当する。



強化学習と脳のメカニズム

- 「脳」の学習メカニズムと類似
- Skinner博士のスキナー箱の実験
  - 仕組み：ラットの飼育ゲージにあるボタンを押すとえさが出てくる
  - ラットが偶然、ボタンに触れる…
    - えさが出てきた！しかしラットにボタンとえさの関係は分からない
  - 偶然の経験を繰り返すと、ボタンを押す（行動）とえさ（報酬）の関係を学習し、行動が強化された



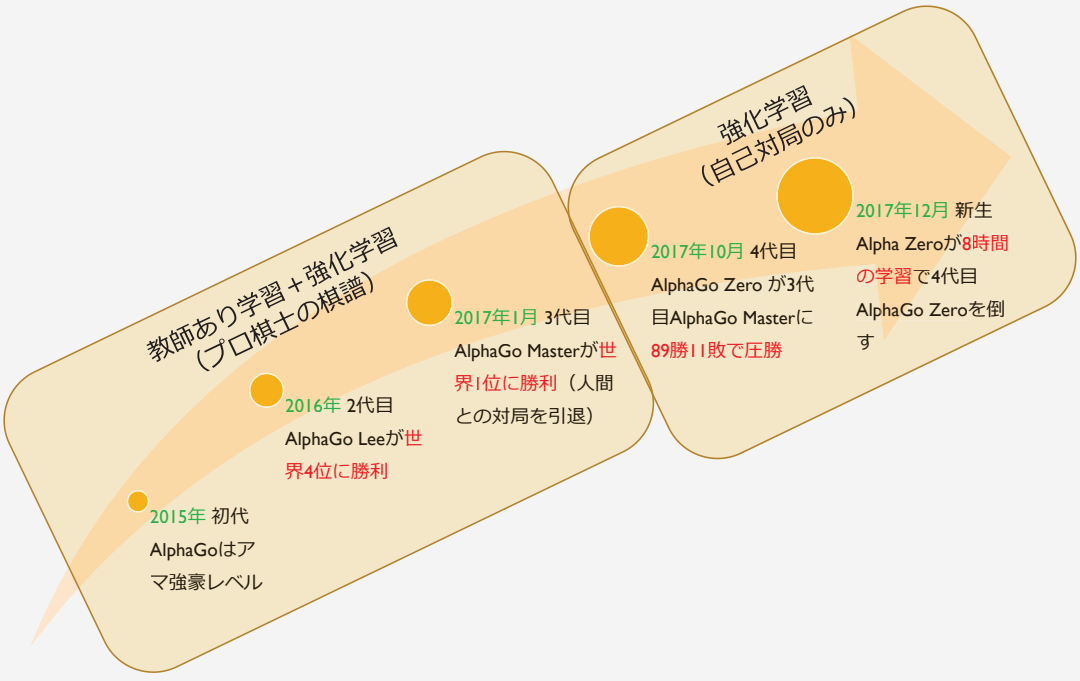
ラットを使用したスキナー博士の実験

過去の強化学習ブームと深層学習

- 1990年代後半から2000年代初頭
  - 強化学習ブームが起こるも、想像した成果は出ず、一度下火になる
- ブーム収束の一つの要因
  - 状態の「縮約表現」が困難だった
- 深層学習と強化学習との出会い
  - 縮約表現(後述)が得意な深層学習
  - “深層強化学習”という分野に発展
  - TVゲームや囲碁で世間に大衝撃！

出典： "reinforcement learning" - Google トレンド
https://trends.google.com/trends/explore?date=all&q=%22reinforcement%20learning%22

囲碁AIで見る強化学習の進化



産業応用

株式会社新潟人工知能研究所
技術開発部

不完全情報ゲームや現実空間へ

- 完全情報ゲーム
  - 両プレイヤーがゲーム状態のすべてを知っている…将棋、囲碁など
- 不完全情報ゲーム
  - ポーカー、麻雀など、誰も分からないような情報が存在するゲーム
- 現実空間でのシステム構築
  - 自動運転技術、高精度工作機械
  - データセンターの冷却システム

麻雀 AI Microsoft Suphx が人間のトッププレイヤーに匹敵する成績を達成

2019年8月29日 | Japan News Center

AI として初めて麻雀 10 段を獲得、技術革新とブレイクスルーを達成

2019 年 6 月、マイクロソフトの AI Microsoft Suphx (Super Phoenix) が、日本のオンライン麻雀対戦プラットフォーム「天鳳」(https://tenhou.net/) において AI として初めて 10 段を達成しました。Suphx は、マイクロソフトの研究開発機関 Microsoft Research Asia (MSRA) が開発した麻雀 AI で、その強さは、最強レベルの人間のプレイヤーに匹敵しています。

5,000 回の対局後、Suphx が天鳳の 10 段を達成

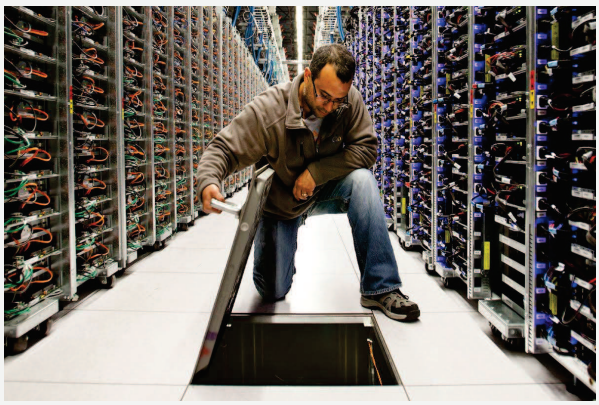
麻雀は、歴史が長く、また幅広い地域で愛好されてきたため、地域によりルールが異なり、共通ルールのもとで長期的な成績を評価するのが難しいゲームです。「天鳳」では、明確な対局ルールと強さを具体的に示す段位システムがあります。さらに、2006 年に開設以来、すべての対戦記録や牌譜 (麻雀の自摸や打牌などの動作、点数の得失などの記録) など、AI の学習にとって有用なデータを公開しており、AI の可能性を検証するのに最適な環境が整っています。

MSRA は、こうした天鳳の特長に着目し、2019 年 3 月、Suphx を天鳳に参加させました。オープンなゲームルームである「特上卓」に参加した Suphx は、今までに 5,000 回以上、人間のプレイヤーと対局を重ね、本年 6 月に AI として初めて 10 段を達成しました。

出典： News Center Japan
「麻雀 AI Microsoft Suphx が人間のトッププレイヤーに匹敵する成績を達成」
https://news.microsoft.com/ja-jp/2019/08/29/190829-mahjong-ai-microsoft-suphx/

現実空間での応用例

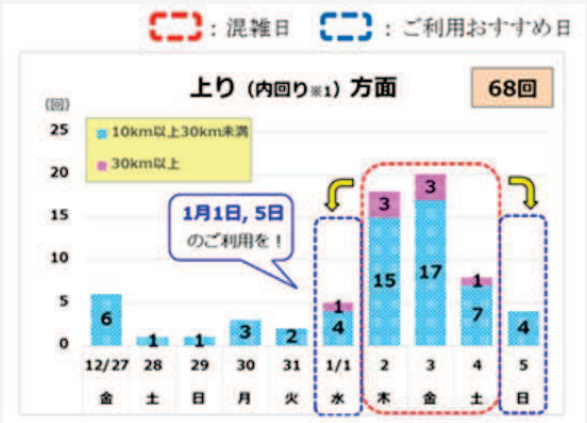
- 例）データセンターの空調
  - どの空調を、どの風向きで、どのタイミングで動かせば、効率的か
  - 現在の室温、稼働中のサーバ、その後のサーバ状況（予測）、外気
  - ただ冷やすわけではなく、どうすれば無駄な冷却を避けられるか？
  - 報酬：省電力で冷却できたときにはプラス、電力消費が多かったり、冷却できなかったときはマイナス



参考： Data centers – Google Data centers
https://www.google.com/about/datacenters/gallery/

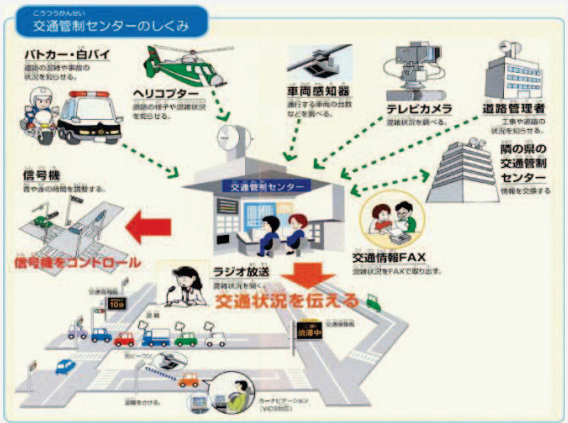
産業応用例：交通システム

教師あり学習の場合…



出典：トラベル Watch「年末年始（2019年～2020年）の渋滞予測、首都圏版」
https://travel.watch.impress.co.jp/docs/news/1219651.html

強化学習の場合…



出典：静岡県警察「交通管制センターってどんなところ？」
http://www.pref.shizuoka.jp/police/anzen/jiko/kiseka/kansecenter/donnatokoro.html

産業応用例：金融トレード

教師あり学習の場合...

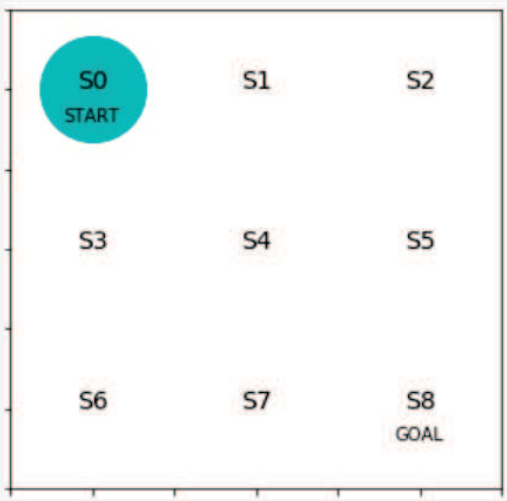
強化学習の場合...

基礎理論

株式会社新潟人工知能研究所
技術開発部

迷路を探索する強化学習

- 迷路課題
  - 3×3のマス目
  - S0…スタート地点
  - S8…ゴール地点
  - 壁は通り抜けられない
- 用語確認
  - 緑の丸… エージェント
  - S0～S8 … S（Status＝状態）



行動(Action)と方策(Policy)

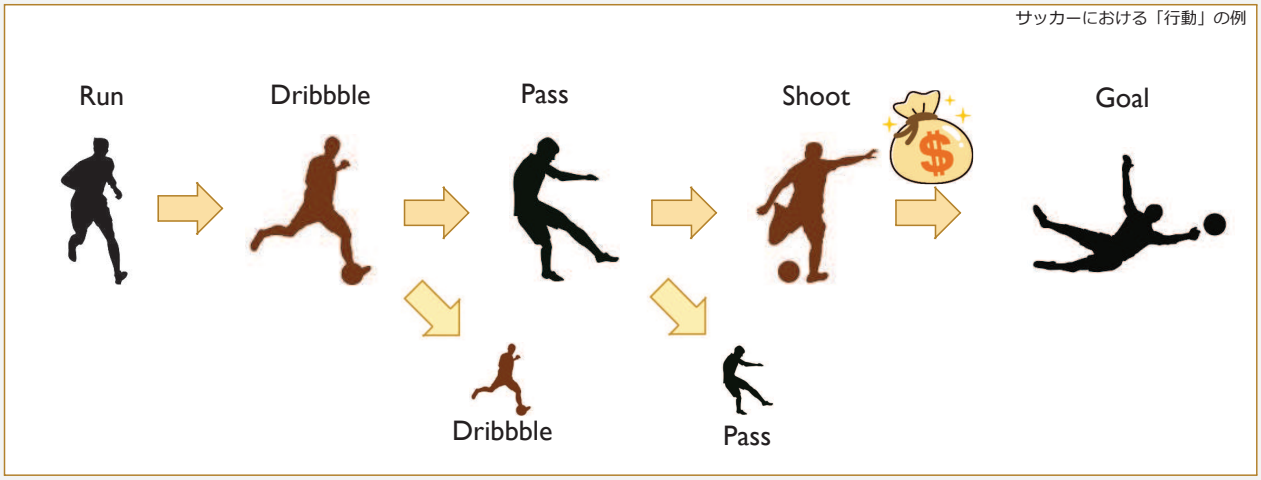
- 行動(a=Action)
  - その状態でエージェントが実行できる行動。例えば状態S0の場合は行動(a)は「右」と「下」の二種類である。
- 方策(p=policy)
  - 状態sと行動aの組み合わせで表現する、エージェントの行動指針ルール。
  - 様々な形式があるが今回は理解しやすい表形式を使用する。表形式は行が状態sを、列が行動aを示し、表の値はその行動を採用する確率を示す。

行動(a)

	行動(上)	行動(右)	行動(下)	行動(左)
0	0.000000	0.500000	0.500000	0.000000
1	0.000000	0.333333	0.333333	0.333333
2	0.000000	0.000000	0.500000	0.500000
3	0.333333	0.333333	0.333333	0.000000
4	0.250000	0.250000	0.250000	0.250000
5	0.333333	0.000000	0.333333	0.333333
6	0.500000	0.500000	0.000000	0.000000
7	0.333333	0.333333	0.000000	0.333333

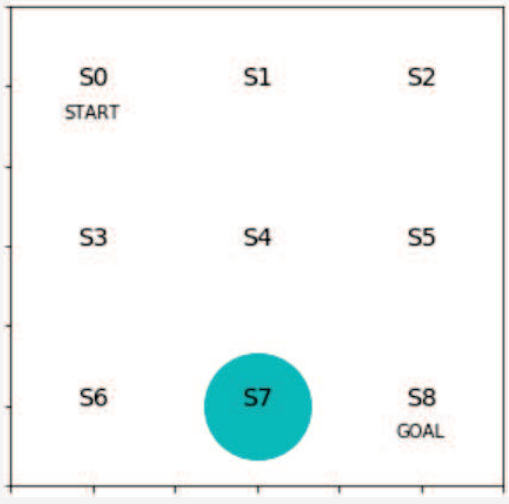
状態(s)　方策(p)

「行動」と「報酬」の例

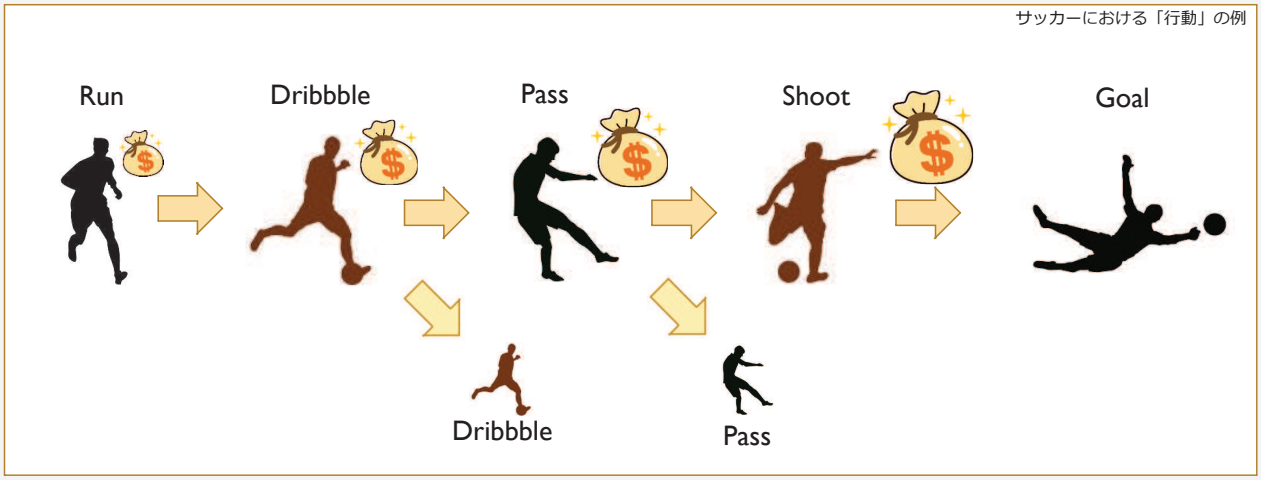


報酬(Reward)と行動価値(Q値)

- いずれもGOALなので報酬をGet
  - 状態s5…下に行けばGOAL(+1点)
  - 状態s7…右に行けばGOAL(+1点)
- ある状況での次の行動の価値を行動価値（＝Q値）で表現する
- しかし、報酬に繋がるS5,S7以外の状態S0やS1やS3では無報酬？
  - そこまでの過程にも報酬を与える
  - Q-Learning 等の手法でQ値を更新

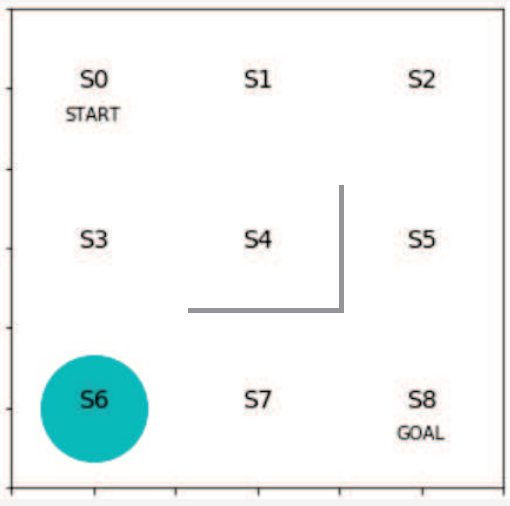


行動価値(Q値)と時間割引(γ(ガンマ))の例



Q-Learningと行動価値(Q値)

- 例えば、状態S7で右に行動することに価値があると分かれば…
- 状態S6では、S7に到達するよう、右に行動する事の価値を上げる
- さらに状態S3では、S4とS6で得られるであろう価値を比較して、下に行動する事の価値を上げる。
- ゴールから逆算して辿ることで、価値を手前に伝搬＝価値反復法

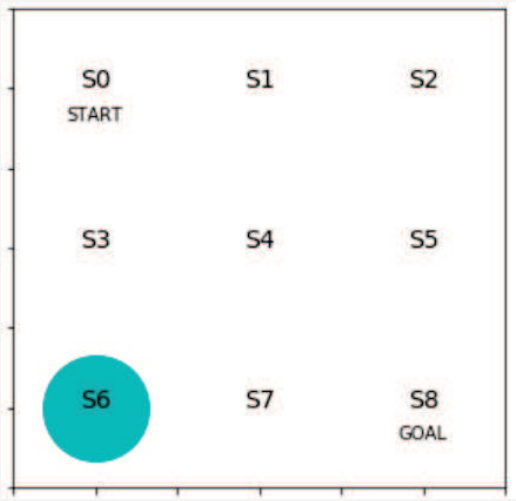


行動価値(Q値)の更新

迷路を解くのにかかったステップ数は4です

	行動(上)	行動(右)	行動(下)	行動(左)
0	0.000000	0.293543	0.413999	0.000000
1	0.000000	0.253959	0.133635	0.434193
2	0.000000	0.000000	0.445081	0.448474
3	0.366831	0.407987	0.509783	0.000000
4	0.311003	0.420637	0.195906	0.417512
5	0.433710	0.000000	0.360868	0.430404
6	0.414262	0.731112	0.000000	0.000000
7	0.880261	0.985013	0.000000	0.088680

20エピソード学習後



ε-greedy法 (イプシロン・グリーディ法)

- 探索と利用のトレードオフ
  - 行動価値（Q値）が最大になる行動を採用するだけだと、最適なQ値が求まっていない状態ではQ値の初期値しだいで行動が確定してしまう
  - 一定確率εでランダムに行動させる。(εは試行数が増えたら小さくする)
  - いつも行動価値の最大行動を利用するのではなく、ときにはランダムに移動させる(探査)させる必要がある

組み合わせ爆発と表形式の限界

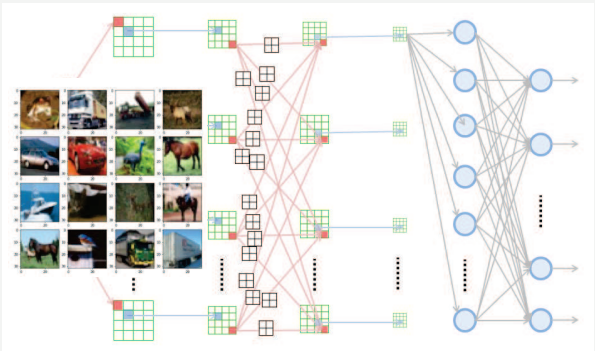
- 表形式＝状態Status(行)×行動Action(列)

- 将棋
  - 状態…自軍と相手の駒（計40枚）の位置
  - 行動…動かす駒と位置（初手でも30種類）
- 自動運転車
  - 状態…センサーの値（アウディA8で23個）
  - 行動…ハンドル、ブレーキ、アクセル

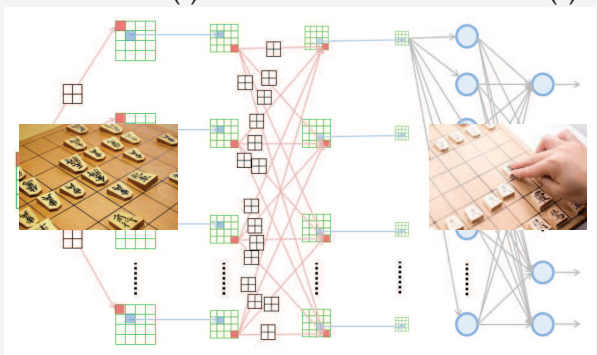
- 膨大な組み合わせになり表形式では無理
  - もとの状態をそこなわないように縮約表現する方法の実現が、強化学習の課題だった

Deep Q-Network（DQN）

- Deep Q-Network（DQN）
  - 深層学習を使ってQ値を推定する



入力層＝状態(s)　　出力層＝行動(a)



強化学習

迷路を探索するプログラムを強化学習で組んでみましょう。今回は代表的な強化学習アルゴリズムであるQ学習（Q-Learning）で、迷路を探索する過程を追ってみます。

ランダムに探索するプログラム

強化学習を組む前に、まずは上下左右にランダムに動きながら、（偶然にでも）ゴールにたどり着くことを目指すプログラムを作成してみます。

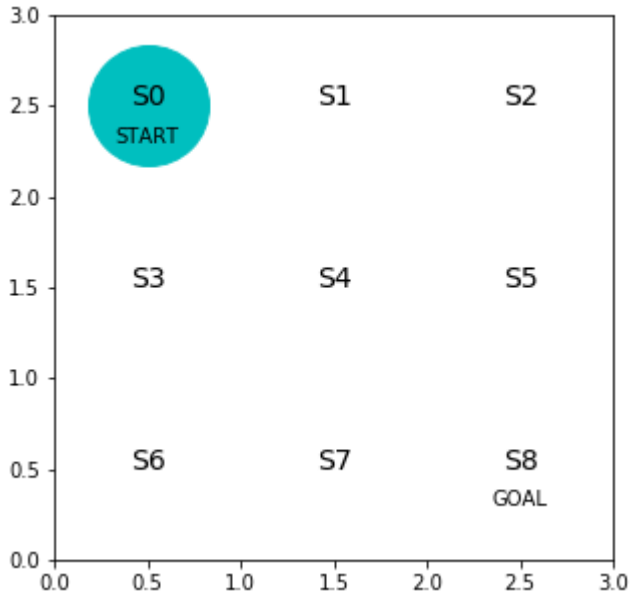
```
# 使用するライブラリのインポート
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

迷路の描画

matplotlibライブラリを利用して、迷路を描画します。四辺以外に、壁を設定することも可能ですが、ここでは問題を簡潔にするために、余計な壁を設置せずに問題に取り組んでみましょう。

```
### 迷路の描画(初期状態) ###

# 図のサイズ・変数名を宣言
fig = plt.figure(figsize=(5, 5))
ax = plt.gca()

# 状態(S0～S8)を描画
plt.text(0.5, 2.5, 'S0', size=14, ha='center')
plt.text(1.5, 2.5, 'S1', size=14, ha='center')
plt.text(2.5, 2.5, 'S2', size=14, ha='center')
plt.text(0.5, 1.5, 'S3', size=14, ha='center')
plt.text(1.5, 1.5, 'S4', size=14, ha='center')
plt.text(2.5, 1.5, 'S5', size=14, ha='center')
plt.text(0.5, 0.5, 'S6', size=14, ha='center')
plt.text(1.5, 0.5, 'S7', size=14, ha='center')
plt.text(2.5, 0.5, 'S8', size=14, ha='center')
plt.text(0.5, 2.3, 'START', ha='center')
plt.text(2.5, 0.3, 'GOAL', ha='center')

# 描画範囲の設定
ax.set_xlim(0, 3)
ax.set_ylim(0, 3)

# 壁を描く（コメントアウト）
# plt.plot([1, 1], [0, 1], color='black', linewidth=2)
```

```
# 現在値(S0)に丸を描画
line, = ax.plot([0.5], [2.5], marker="o", color='c', markersize=60)
```



方策の元となるθを初期化

ここでは、状態(s=Status)における、行動(a=Action)の組み合わせを表現した、「方策(p=Policy)」を作成します。

まず、方策(p)のもとになるパラメータθ(シータ）を初期化します。θはシンプルに、壁のある方向に0、進める方向に1を入力した、np.array形式で定義します。

```
# 初期の方策(Policy)を決定するパラメータthetaを設定する

# 行は状態(s)、列は時計回り(↑、→、↓、←)に移動可能=1、移動不可=0
theta = np.array([  [0, 1, 1, 0],  # s0
                    [0, 1, 1, 1],  # s1
                    [0, 0, 1, 1],  # s2
                    [1, 1, 1, 0],  # s3
                    [1, 1, 1, 1],  # s4
                    [1, 0, 1, 1],  # s5
                    [1, 1, 0, 0],  # s6
                    [1, 1, 0, 1],  # s7
                    ])             # s8はゴールなので方策なし

pd.DataFrame(theta, columns=['行動(上)','行動(右)','行動(下)','行動(左)'] )
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	行動(上)	行動(右)	行動(下)	行動(左)
0	0	1	1	0
1	0	1	1	1
2	0	0	1	1
3	1	1	1	0
4	1	1	1	1
5	1	0	1	1
6	1	1	0	0
7	1	1	0	1

方策(p)の算出

続いて、このパラメータθを変換して方策(p)を求めます。今回は単純な変換方法を採用し、進める方向に対してθの値を割合に変換して確率とします。

```
# 方策パラメータθ(theta)を、行動方策(policy)に変換

[m, n] = theta.shape
policy_table = np.zeros((m, n))

for i in range(0, m):
    policy_table[i, :] = theta[i, :] / np.nansum(theta[i, :])  # 割合計算

pd.DataFrame(policy_table, columns=['行動(上)','行動(右)','行動(下)','行動(左)'] )
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	行動(上)	行動(右)	行動(下)	行動(左)
0	0.000000	0.500000	0.500000	0.000000
1	0.000000	0.333333	0.333333	0.333333

	行動(上)	行動(右)	行動(下)	行動(左)
2	0.000000	0.000000	0.500000	0.500000
3	0.333333	0.333333	0.333333	0.000000
4	0.250000	0.250000	0.250000	0.250000
5	0.333333	0.000000	0.333333	0.333333
6	0.500000	0.500000	0.000000	0.000000
7	0.333333	0.333333	0.000000	0.333333

注釈: 行動(上右下左)する確率を、S(状態)ごとに表現した、表が出来上がりました。例えば、スタート地点(S0)では、右と下に50%の確率で移動することが分かります。

移動後の状態sを求める

ここからは迷路探索に必要な関数を作成します。強化学習の理解とは直接関係のない部分なので、プログラムの詳細まで理解する必要はありませんが、実際に手を動かしてみて、各関数の役割を体験してみましょう。

まずは、1つ移動した後に、エージェントがどの位置にいるのか（次の状態sは何番か？）を把握する関数を作成します。

```
### 1つ移動した後の状態sを求める関数 ###
 def goto_next(policy_table, s):

    direction = ["up", "right", "down", "left"]

    # policy_table[s,:]の確率に従って、directionが選択される
    next_direction = np.random.choice(direction, p=policy_table[s, :])

    if next_direction == "up":
        s_next = s - 3  # 上に移動するときは状態の数字が3小さくなる
    elif next_direction == "right":
        s_next = s + 1  # 右に移動するときは状態の数字が1大きくなる
    elif next_direction == "down":
        s_next = s + 3  # 下に移動するときは状態の数字が3大きくなる
    elif next_direction == "left":
        s_next = s - 1  # 左に移動するときは状態の数字が1小さくなる

    return s_next
```

```
# goto_next関数の動作チェック
goto_next( policy_table , 0 )
```

```
1
```

注釈: goto_next関数の動作チェックを行います。先程作成しておいた、行動方策の表（pi_table）と、スタート地点の状態(S0の0)を、goto_next関数に引き渡すと、1か3の値が戻ってきます。

これはスタート地点からは右か下にしかいけませんので、右に向かった場合は、状態がS1になるので1が、下に向かった場合は、状態がS3になるので3が戻ってくるからです。

0以外の状態からgoto_nextを実施しても、想定通りに正しく機能するか（四方の壁を突き破るようなことはないか）実験してみましょう。

迷路に挑戦（ランダムウォーク）

それではさっそくエージェントに（ランダムですが）迷路に挑戦してもらいましょう！

```
# 迷路に挑戦

s = 0  # スタート地点
state_history = [0]  # エージェントの移動を記録するリスト

# ゴールするまで永久ループ（本来はmaxループ回数を設定すべきだが割愛）
while (1):
    next_s = goto_next(policy_table, s)  # 行動：goto_next関数で次の状態(next_s)を取得する
    state_history.append(next_s)         # 記録：リストに次の状態(エージェントの位置)を記録

    # 先程得たnext_sを新しいs(状態)とする
    s = next_s

    # ゴール地点ならループを強制脱出
    if next_s == 8:
        break

print(state_history)
print("迷路を解くのにかかったステップ数は"+str(len(state_history)-1)+"です")
```

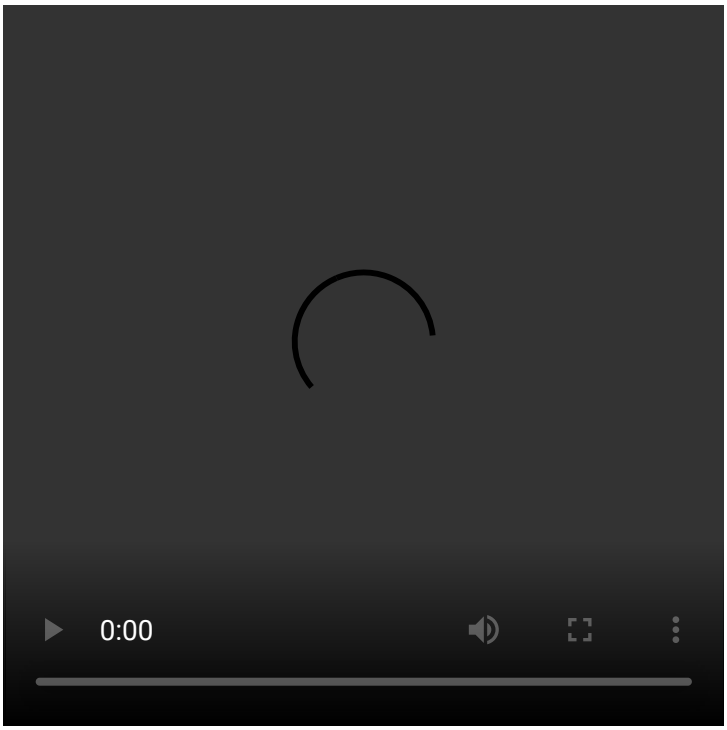
```
[0, 1, 0, 3, 0, 3, 6, 3, 6, 3, 6, 3, 0, 3, 0, 3, 4, 3, 0, 3, 4, 3, 6, 7, 8]
迷路を解くのにかかったステップ数は24です
```

注釈: 実行するごとに迷路をとくステップ数が違うと思います。

ランダムウォークの描画

ではせっかくですので、ランダムウォークの様子を可視化してみましょう。できるだけウロウロしているところが見たいので、上のセルでステップ数が10を超してから、下のセルを実行してください。

```
# エージェントの移動の様子を可視化
from matplotlib import animation
from IPython.display import HTML

# 背景画像の初期化
def init():
    line.set_data([], [])
    return (line,)

# 描画内容の生成(フレーム単位)
def animate(i):
    state = state_history[i]  # 現在の場所を描く
    x = (state % 3) + 0.5     # 状態のx座標は、3で割った余り+0.5
    y = 2.5 - int(state / 3)  # y座標は3で割った商を2.5から引く
    line.set_data(x, y)
    return (line,)

# 描画関数を用いて動画生成
anim = animation.FuncAnimation(fig, animate, init_func=init,
frames=len(state_history), interval=500, repeat=False)

# 動画をHTMLとして描画
HTML(anim.to_html5_video())
```



注釈: 行が状態sを、列が行動aを示す行動価値関数Q(s,a)を表形式で実装します。最初は正しい価値の値が分からないのでランダムな値を与えます。

Q-Learningによる探索

それでは、Q-Learningによる探索を始めます。まず、迷路をプレイするために、次のコマに移動するためのget_action関数を用意しておきます。

ε-greedy法

さらにここではε-greedy法によって、ときおりQ値に従わないで、適当な探索をしてしまうように仕込んでおきます。

```
# ε-greedy法を実装（イプシロン・グリーディ法）

# s         = 状態
# Q         = 行動価値関数Qの表
# epsilon   = ε-greedy法の初期値

def get_action(s, Q, epsilon, policy_table):

    direction = ["up", "right", "down", "left"]

    # 行動を決める
    if np.random.rand() < epsilon:
        # εの確率でランダムに動く
        next_direction = np.random.choice(direction, p=policy_table[s, :])
        print('探索のためランダムに動いてみた')
    else:
        # 行動価値関数Qの最大値の行動を採用する
        next_direction = direction[np.nanargmax( Q[s, :] )]
        print('行動価値(Q値)に従って動いてみた')

    # 決めた行動で次の状態を決める
    if next_direction == "up":
        action = 0
        s_next = s - 3  # 上に移動するときは状態の数字が3小さくなる
    elif next_direction == "right":
        action = 1
        s_next = s + 1  # 右に移動するときは状態の数字が1大きくなる
    elif next_direction == "down":
        action = 2
        s_next = s + 3  # 下に移動するときは状態の数字が3大きくなる
    elif next_direction == "left":
        action = 3
        s_next = s - 1  # 左に移動するときは状態の数字が1小さくなる

    return [action, s_next]
```

get_action関数の動作チェックを行います。 εを変更すると、探索のためにランダムに動くときと、行動価値(Q値)に従って動くときの割合を変更することができます。

```
s = 4           # スタート地点
epsilon  = 0.1  # ε-greedy法の初期値

# get_action関数の動作チェック
[a, s_next] = get_action(s, theta, epsilon, policy_table)
s_next
```

行動価値(Q値)に従って動いてみた

1

行動価値(Q値)の更新(準備)

次に、行動と報酬に応じて行動価値(Q値)を更新するQ_learning関数を準備します。この関数は後ほど使用します。ここでは数式の説明は省略しますが、数学がお得意なかたは関連資料などで数式を追ってみてください。

```
# Q学習による行動価値関数Qの更新

# s(状態)、a(行動)、r(報酬)、Q_table(Q値の表)
# gakushu  = 学習率(etaη)
# waribiki = 時間割引率(gammaγ)

def Q_learning(s, a, r, s_next, Q_table, gakushu, waribiki):

    if s_next == 8:  # ゴールした場合
        Q_table[s, a] = Q_table[s, a] + gakushu * (r - Q_table[s, a])

    else:
        Q_table[s, a] = Q_table[s, a] + gakushu * (r + waribiki *
np.nanmax(Q_table[s_next,: ]) - Q_table[s, a])

    return Q_table
```

迷路を解く関数(準備)

次に、迷路のスタートからゴールするまでをループで繰り返すためのtry_maze関数を準備します。

```
# Q学習で迷路を解く関数

# epsilon  = ε-greedy法の初期値
# gakushu  = 学習率(etaη)
# waribiki = 時間割引率(gammaγ)

def try_maze(Q_table, epsilon, gakushu, waribiki, policy_table):

    s    = 0   # 状態(S0=スタート地点)
    step = 0   # ゴールまでのステップ数
```

```
    # ゴールするまでループする
    while ( s != 8 ):

        step = step + 1

        [a, s_next] = get_action(s, Q_table, epsilon, policy_table)

        # ゴールに着いたら報酬を与える
        if s_next == 8:
            r = 1
        else:
            r = 0

        # 価値関数を更新
        Q_table = Q_learning(s, a, r, s_next, Q_table, gakushu, waribiki)

        s = s_next

    # 戻り値:ステップ数,Q値
    return [step, Q_table]
```

補足: ここでは迷路に挑戦するためのtry_maze関数を定義しているだけなので、実行しても何も起きません。このあと、try_maze関数を実行しましょう。

初期パラメーターの設定

ε-greedy、学習率、時間割引率のパラメーターを設定します。 また、行動価値関数(Q値)をθをベースにして作成しますが、ここではQ値の初期値はランダムにしてしまいます。

```
# Q学習で迷路を解くためのパラメータ初期化

epsilon  = 0.5  # ε-greedy法の初期値
gakushu  = 0.1  # 学習率(etaη)
waribiki = 0.9  # 時間割引率(gammaγ)

# 行と列の数をa, bに格納
[a, b] = theta.shape

# 初期の行動価値関数Qを表で設定
Q_table = np.random.rand(a, b) * theta

display( pd.DataFrame( theta  , columns=['行動(上)','行動(右)','行動(下)','行動
(左)']  ) )
display( pd.DataFrame( Q_table, columns=['行動(上)','行動(右)','行動(下)','行動
(左)']  ) )
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

	行動(上)	行動(右)	行動(下)	行動(左)
0	0	1	1	0
1	0	1	1	1
2	0	0	1	1
3	1	1	1	0
4	1	1	1	1
5	1	0	1	1
6	1	1	0	0
7	1	1	0	1

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	行動(上)	行動(右)	行動(下)	行動(左)
0	0.000000	0.381857	0.896516	0.000000
1	0.000000	0.144095	0.941308	0.759527
2	0.000000	0.000000	0.980440	0.960771
3	0.030691	0.312031	0.272073	0.000000
4	0.693175	0.950381	0.014353	0.548878
5	0.728419	0.000000	0.314714	0.571714
6	0.732386	0.872582	0.000000	0.000000
7	0.066843	0.226004	0.000000	0.927796

注釈: 上段がθのテーブルで、下段が行動価値(Q値)のテーブルです。θを元にして、ランダム性をもたせたQ値が生成できました。

迷路に挑戦！

おまたせしました、いよいよ迷路に挑戦しましょう。最初はすごいステップがかかる場合もあれば、いきなり少ないステップでクリアできてしまう場合もあるかと思います。

これは行動価値(Q値)がランダムですので、たまたま迷路にフィットしたQ値が生成されていると、いきなり賢い（ような気がする）エージェントになります。

```
# Q学習で迷路を解き、移動した履歴と更新したQを求める
[step, Q] = try_maze(Q_table, epsilon, gakushu, waribiki, policy_table)

# ε-greedyを少しずつ小さくする（最初は冒険するが、徐々に安全志向になっていく）
epsilon = epsilon / 2

print("迷路を解くのにかかったステップ数は" + str(step) + "です")
pd.DataFrame( Q_table, columns=['行動(上)','行動(右)','行動(下)','行動(左)']  )
```

```
探索のためランダムに動いてみた
行動価値(Q値)に従って動いてみた
行動価値(Q値)に従って動いてみた
行動価値(Q値)に従って動いてみた
行動価値(Q値)に従って動いてみた
探索のためランダムに動いてみた
迷路を解くのにかかったステップ数は6です
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	行動(上)	行動(右)	行動(下)	行動(左)
0	0.000000	0.381857	0.834947	0.000000
1	0.000000	0.144095	0.941308	0.759527
2	0.000000	0.000000	0.949340	0.960771
3	0.030691	0.366362	0.272073	0.000000
4	0.693175	0.920900	0.014353	0.548878
5	0.743817	0.000000	0.383243	0.571714
6	0.732386	0.872582	0.000000	0.000000

	行動(上)	行動(右)	行動(下)	行動(左)
7	0.066843	0.226004	0.000000	0.927796

補足: Ctrl+Enterで、何度かQ学習を実行しているうちに、ステップ数が4に収束していきます。また、何十回と繰り返すうちに、Q値(行動価値関数)も最適化されていきます。ゴール直前の状態S7やS5の行が直感的に理解しやすいと思います。

なお、もう一度、未学習の状態からやり直したい場合は、ひとつ上のセル「# Q学習で迷路を解くためのパラメータ初期化」から、実行しなおしてください。

AI2-7

AI クラウドによるシステム構築演習

－ 講義内容 －

- クラウドサービスの概要
- 事例紹介
- 演習： SONY Neural Network Console で画像分類

AI 2-7
AIクラウドによるシステム構築演習

令和元年度厚生労働省　教育訓練プログラム開発事業　AI講座

2019年12月3日（火）PM

モリパワー株式会社　鬼島佳子

AIシステムの構成

クラウドサービスの利用形態

- インフラサービスを利用する（Infrastructure as a Service：IaaS）
  いわゆるクラウドコンピューティング
  クラウド上のリソース（サーバー、GPU、ストレージ、ネットワーク…）を利用
- 訓練済みモデルのAPIサービスを利用する
  事前に訓練されたモデルで推論を行う
  画像認識、OCR、音声認識、翻訳 …
- 独自モデルを作成するサービスを利用する（≒Platform as a Service：PaaS）
  クラウド上に訓練データをアップロードしてモデルを作成・利用

クラウドサービス利用の利点

- 手軽さ
  クリックだけでインフラ構築やモデル作成ができる
- 開発工期の短縮
  ありものを利用するので、一から開発するよりも速い
- 独自でモデル構築するよりも高精度の結果が得られる
  大量データによる学習済みモデルや、大手ベンダーの技術による高精度のモデルが利用できる

クラウドサービス利用の欠点

- コスト

  月額制・従量課金制などで常にランニングコストがかかる

- 特殊な課題では利用できないことも

  クラウド上のモデル構築サービスを利用したとしても、精度が出ないことがある

  例）Google Visionは実在するものの画像認識には強いが、アニメ絵のようなものは苦手

IaaS（Infrastructure as a Service）

AI開発で必要なインフラ

- 大量データを扱うためのストレージ容量
- GPU
- （サービス提供の場合）ネットワーク、拡張性、負荷分散、セキュリティ…

主なサービス

- Google Cloud Platform（GCP）
- Microsoft Azure
- IBM Cloud
- Amazon Web Service（AWS）
- Alibaba Cloud　※中国・アジア向け

非機能要件とコストで考えるインフラ

	オンプレミス	プライベート クラウド	パブリック クラウド
費用	高	中	低
拡張性	低（お金をかければ高）	中	高
障害対応	自社	業者任せ	業者任せ
セキュリティ	高	中	低

コスト・リスクをどこまで許容するか、ユーザの合意を得る

クラウドのセキュリティ

問題点

- 通信路は暗号化されない
- クラウドにアップしたデータはホスティング業者は自由に利用できる（特にPaaS）
  - ⇒ モデル、個人情報、営業秘密、機微（センシティブ）情報、ノウハウ…

対策

- 最低でもプライベートクラウド
- VPNが利用できるサービスを使う
- 生データはアップロードしない。特徴量（数値）に加工してアップロード
  - 生データ保存・加工用にオンプレミスのストレージ・python環境もある程度必要

APIサービス

訓練済みモデルで推論する（画像認識、OCR、音声認識、翻訳 …）
リクエスト数に対する従量課金制が多い（1,000リクエストあたり約164円 など）

主なサービス

- Google Cloud AI ビルディング ブロック
  https://cloud.google.com/products/ai/building-blocks/
- Microsoft Azure Cognitive Services
  https://azure.microsoft.com/ja-jp/services/cognitive-services/
- IBM Watson API
  https://www.ibm.com/watson/jp-ja/developercloud/services-catalog.html
- Amazon AIサービス群
  https://aws.amazon.com/jp/machine-learning/

参考：AI系APIサービス機能比較（画像） - Qiita
https://qiita.com/yuxzux/items/c1abbc83c8b681530b23

PaaS（Platform as a Service）

インターネット上でアプリケーションを構築・稼働させるプラットフォームを提供
クラウド上で独自モデルを構築
サービスによっては構築したモデルのAPIサービス化や周辺機能の提供まで行う

主なサービス

- Google AutoML（ベータ版）
  https://cloud.google.com/automl/?hl=ja
- Microsoft Azure Machine Learning
  https://azure.microsoft.com/ja-jp/services/machine-learning/
- Amazon SageMaker
  https://aws.amazon.com/jp/sagemaker/
- IBM Watson
  https://www.ibm.com/watson/jp-ja/

Google AutoML 事例

株式会社LIFULL

掲載物件数ナンバーワンの不動産・住宅情報サイト「LIFULL HOME'S」を運営

- 「LIFULL HOME'S」に掲載する物件画像の分類にAutoMLを利用（トイレ・キッチン・バス…）
  - 自社の入稿ツールにAutoMLを組み込んだ
  - 画像を一括アップロードすると分類を自動選択
- 手作業で分類入力するよりも作業時間が短縮（40～50秒 → 10～12秒）
- 分類精度の向上（3.9%改善）
- TensorFlowで実装した独自モデルよりも高精度だった
  - Googleの最新技術によるモデル構築

https://cloud.google.com/blog/ja/topics/customers/automl-lifull

クラウド破産に注意

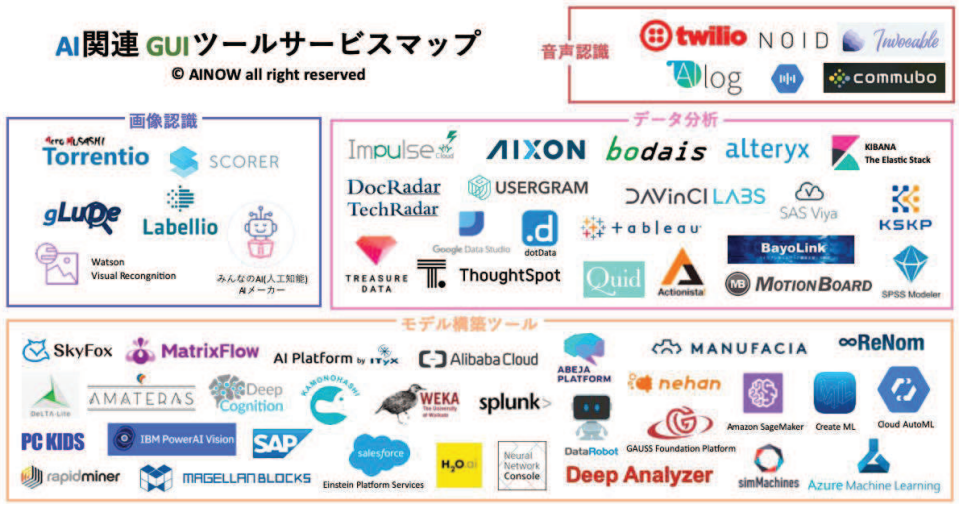
クラウド破産（クラウド死）とは…
クラウドサービスの利用料金が意図せず高額になること

友人A：

「初めてのAIプロジェクトで、学習にどれくらい時間がかかるか見当がつかず、とりあえず回したら400万円請求されました」

- クラウドの利用料金も事前に見積もること
- 利用料金のアラート（警告）設定を忘れずに
- 学習を途中で中止した場合、料金だけ発生してモデルが手元に残らないサービスもあるので注意

GUIツール（not クラウド含む）



出典：プログラミング不要！ 約50のAI構築GUIツールをまとめたサービスマップを公開！ | AI専門ニュースメディア AINOW　https://ainow.ai/2019/07/09/173221/

モデル構築演習

SONY Neural Network Console を使って、画像を分類するモデルを構築してみましょう。

https://dl.sony.com/ja/

SONY Neural Network Console で画像分類

手書き文字の「4」と「9」の2種類の画像を分類してみましょう。

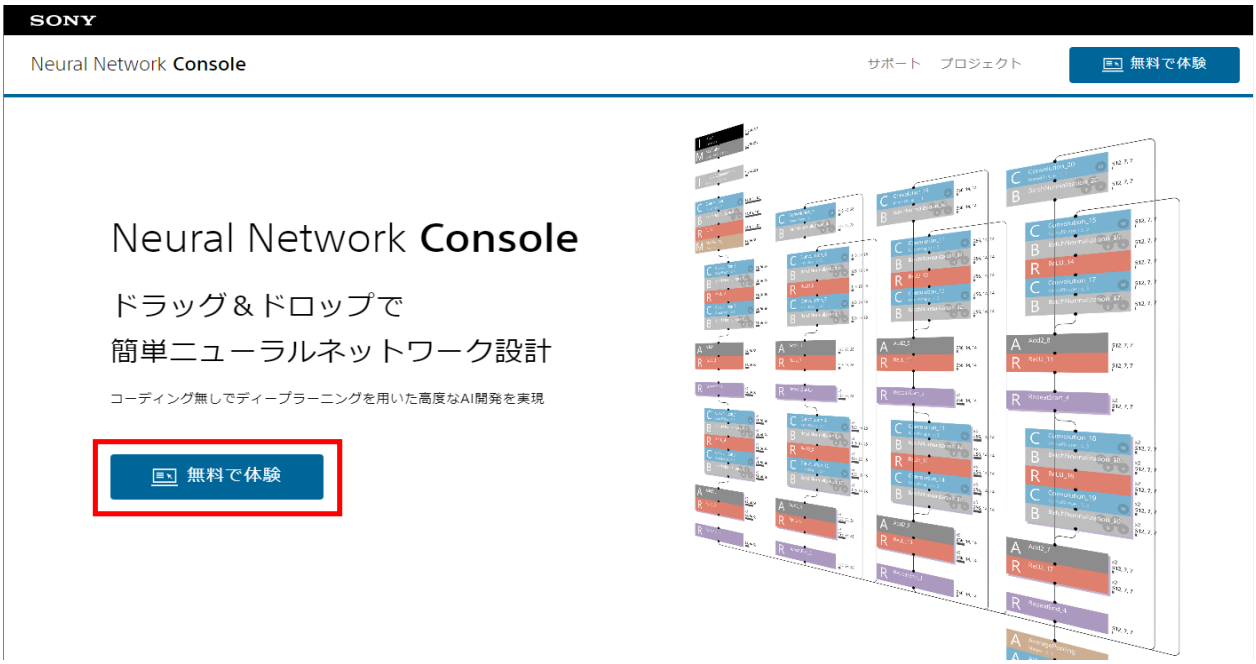
※この演習の内容はサンプルプロジェクト「tutorial.basics.01_logistic_regression」と同じですが、
サンプルプロジェクトは使わず、一から作成してみましょう。


利用の開始

SONY Neural Network Console にアクセスします。

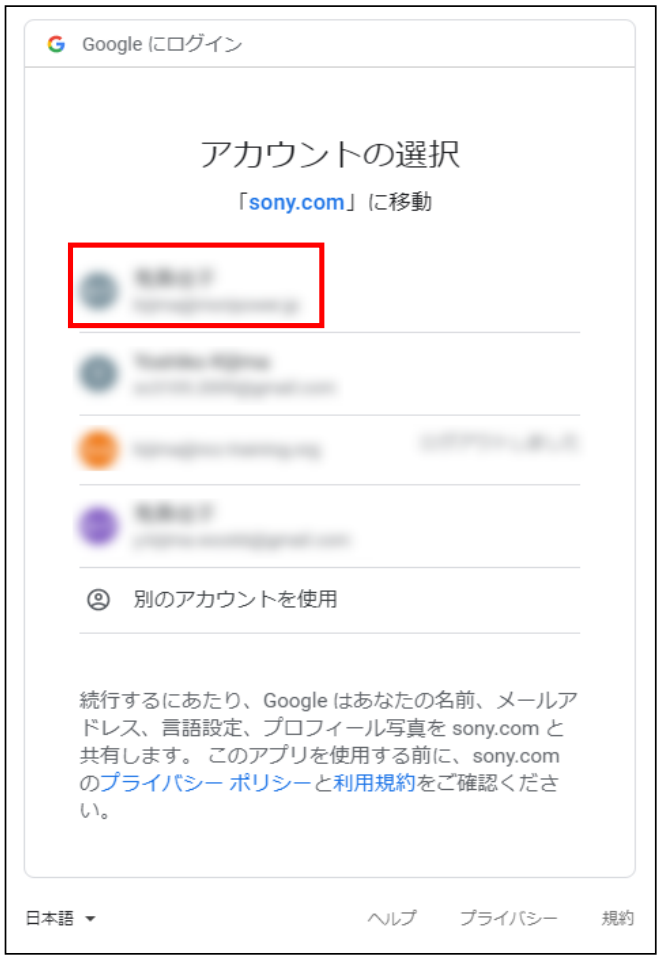
https://dl.sony.com/ja/

「無料で体験」をクリックします。

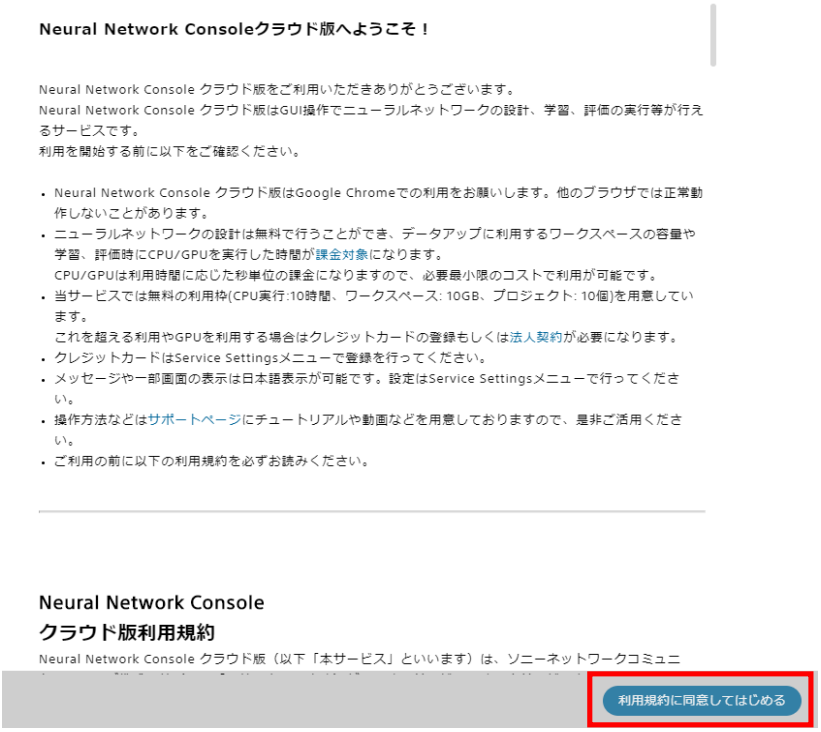


サインインアカウントの選択でGoogleをクリックします。

使用するアカウントを選択します。

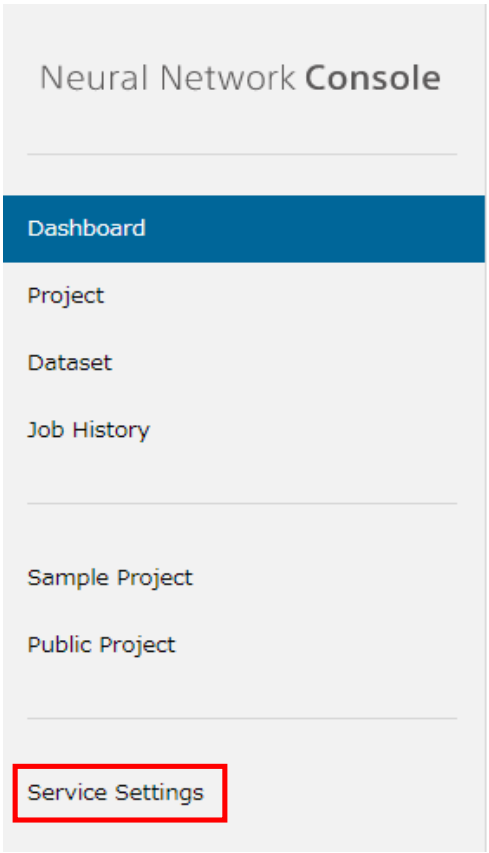


利用規約が表示されたら、「利用規約に同意してはじめる」をクリックします。



日本語化

Service Settings メニューをクリックします。

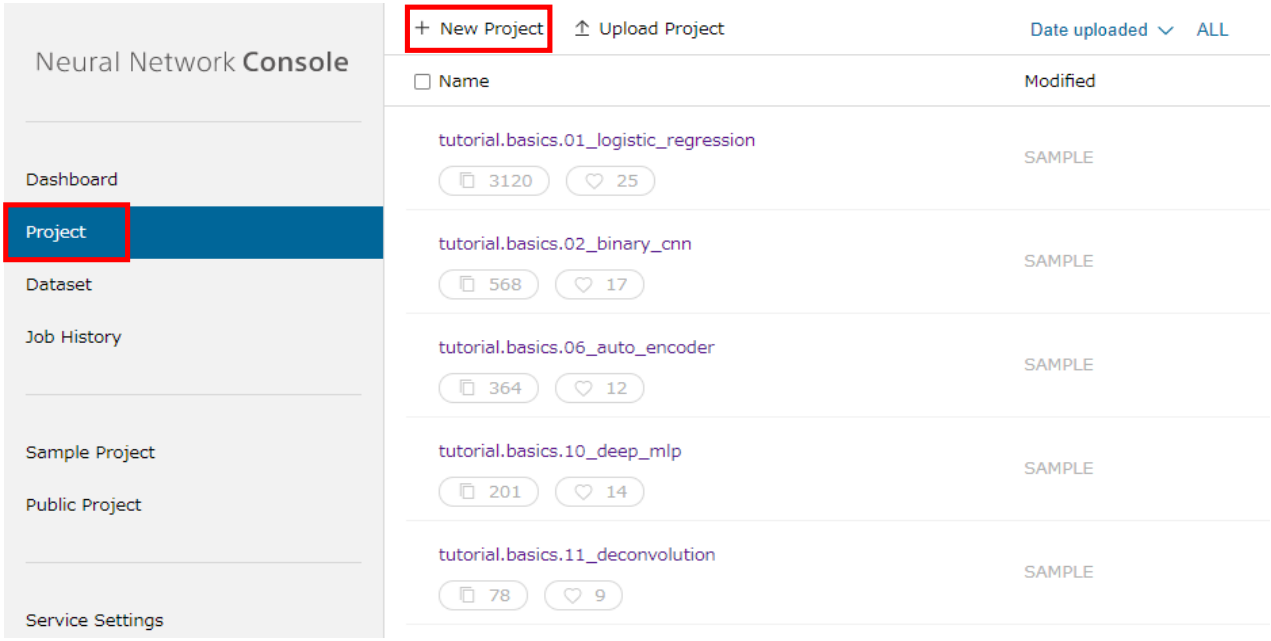


言語を日本語に変更します。

プロジェクトの作成

project メニューをクリックします。

続いて、「New Project」をクリックします。

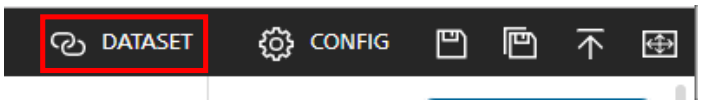


プロジェクト名に「4or9」と入力し、「OK」をクリックします。

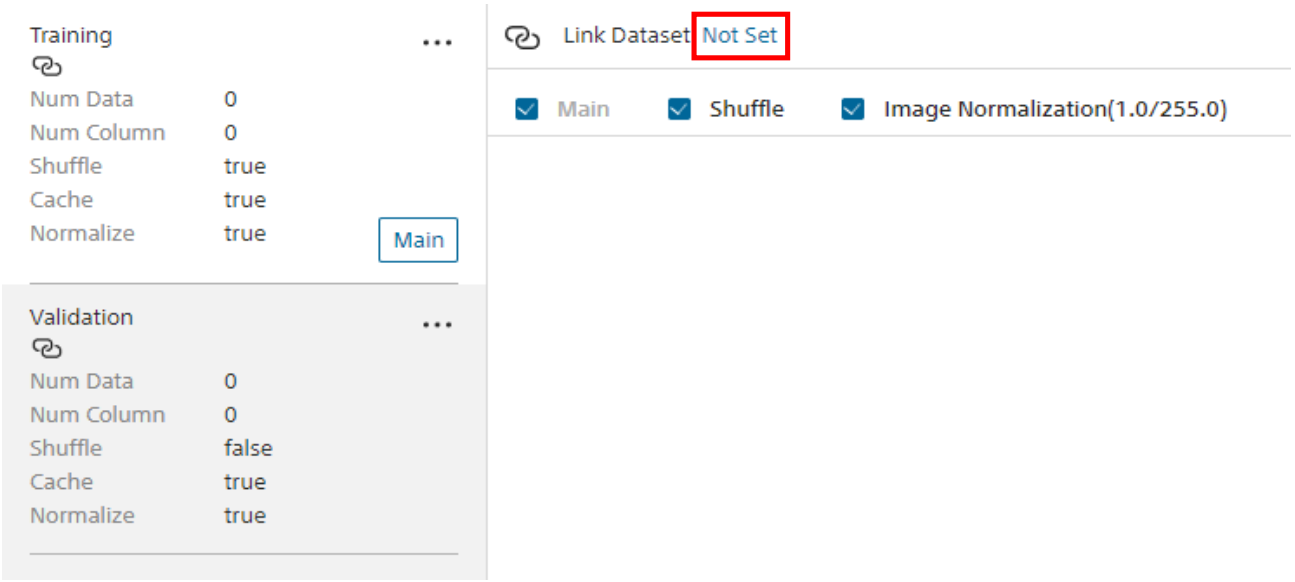
データセットの設定

学習と評価に使用するデータセットを設定します。

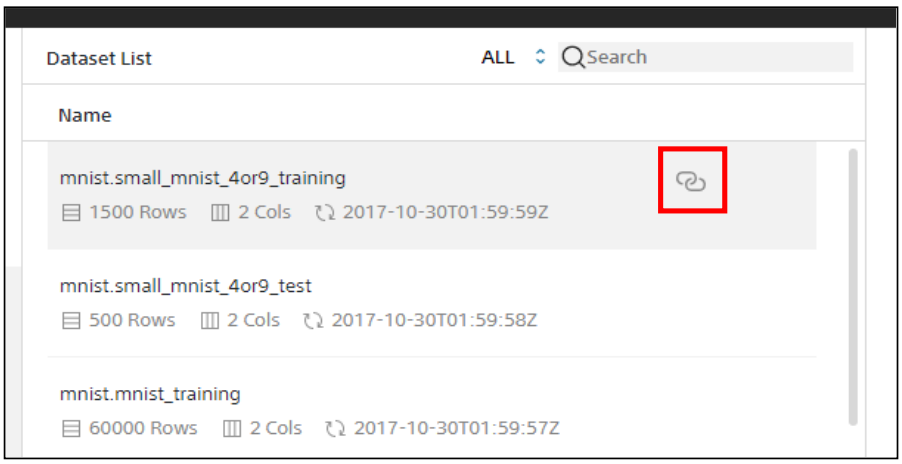
右上の DATASET メニューをクリックします。



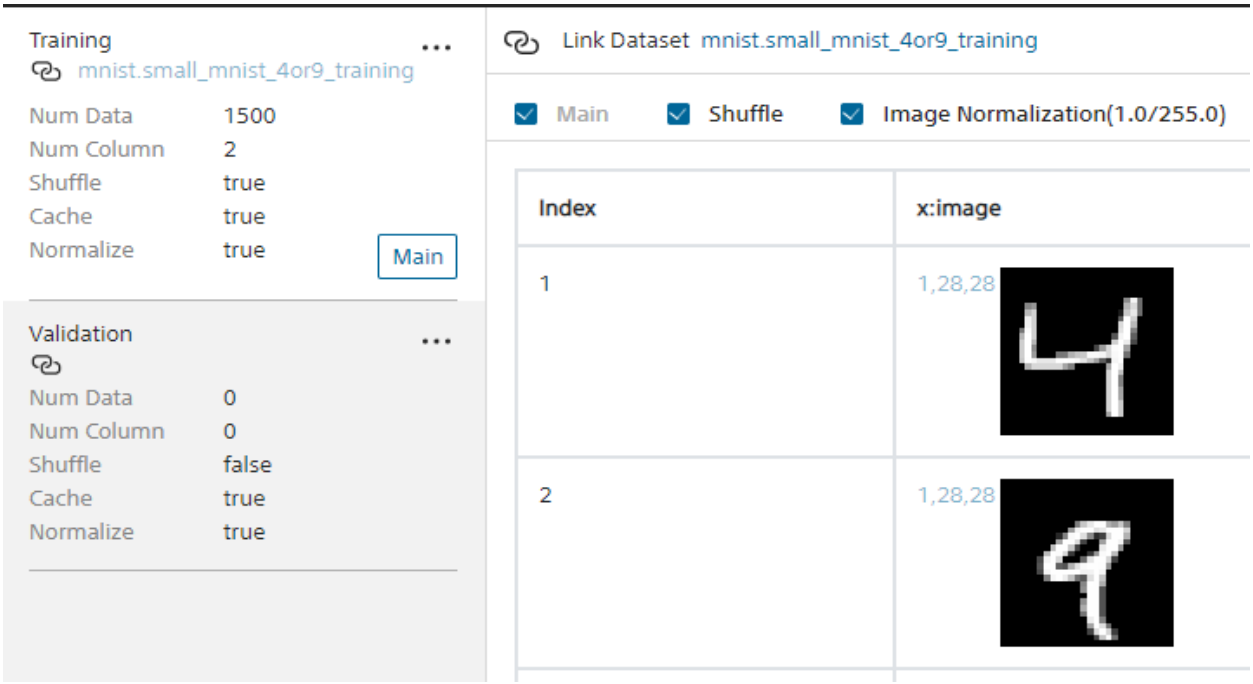
左側のメニューでTrainingが選択されている（白背景になっている）ことを確認し、
「Not Set」をクリックします。



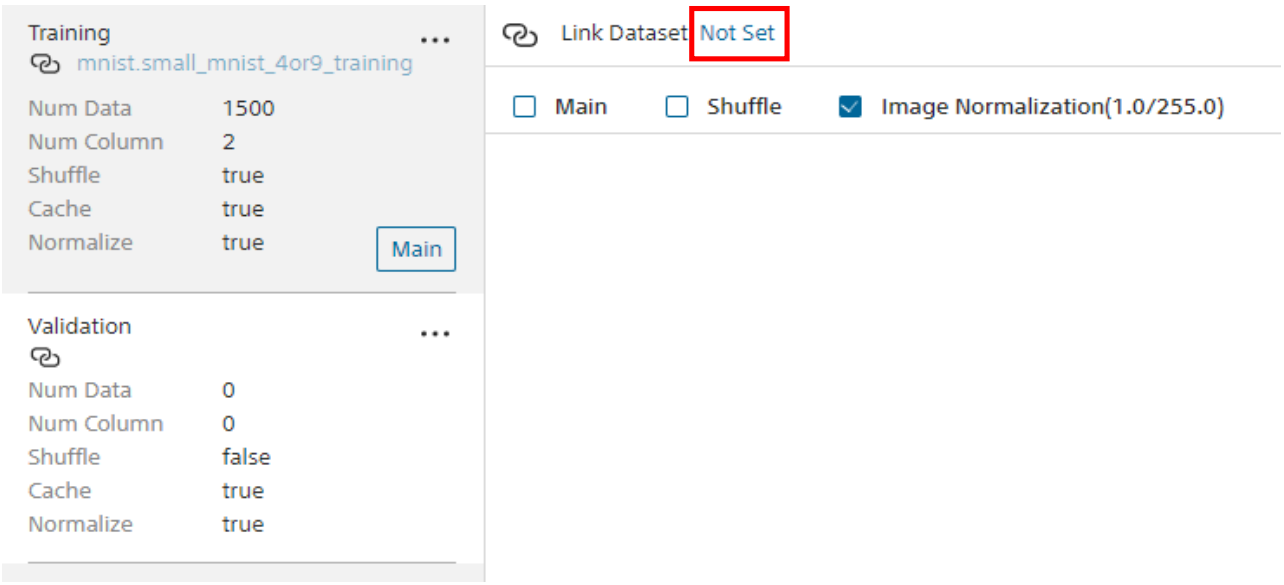
Dataset List から「mnist.small_mnist_4or9_training」のリンクマークをクリックします。



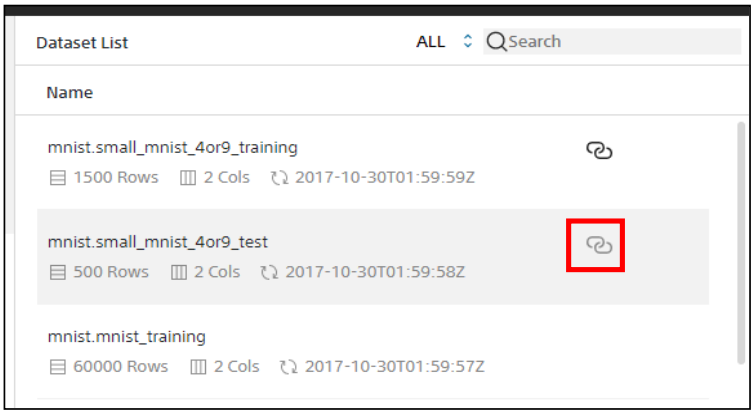
データセットがリンクされました。



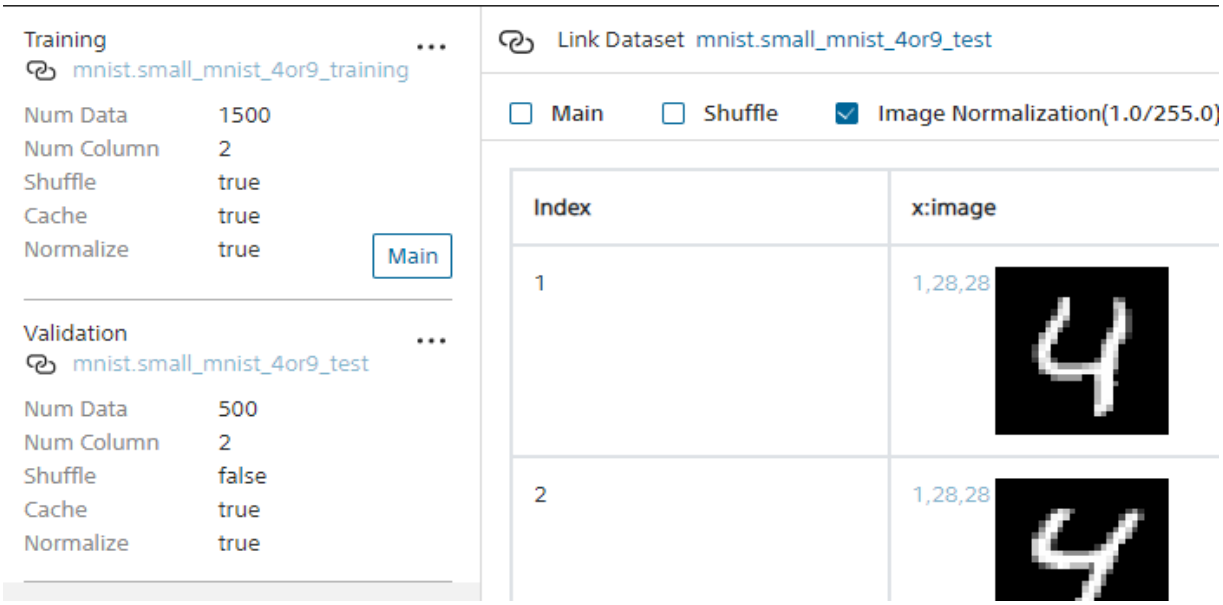
左側メニューでValidationを選択し、「Not Set」をクリックします。



Dataset List から「mnist.small_mnist_4or9_test」のリンクマークをクリックします。

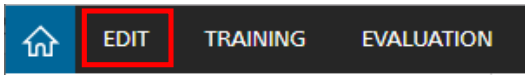


データセットがリンクされました。



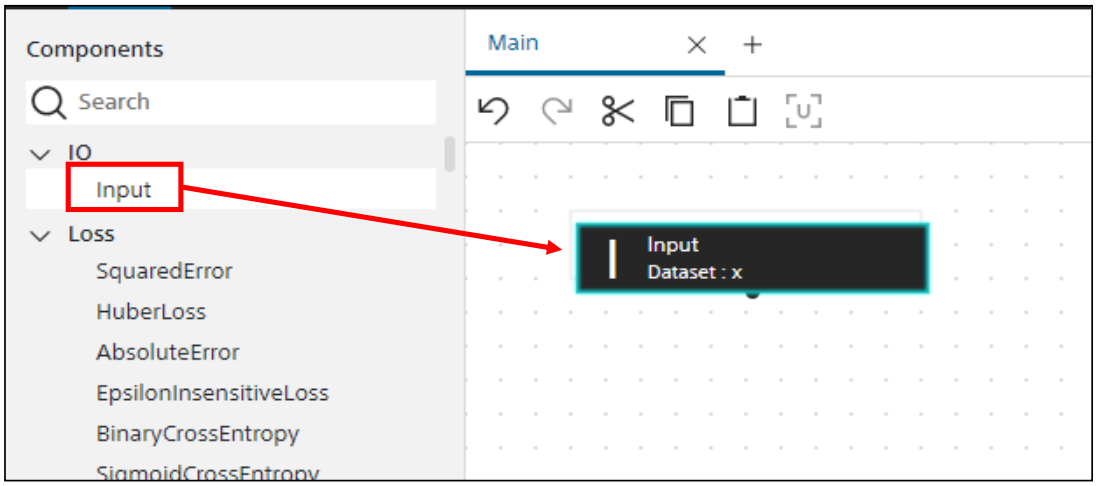
モデルの定義

左上の「EDIT」をクリックします。

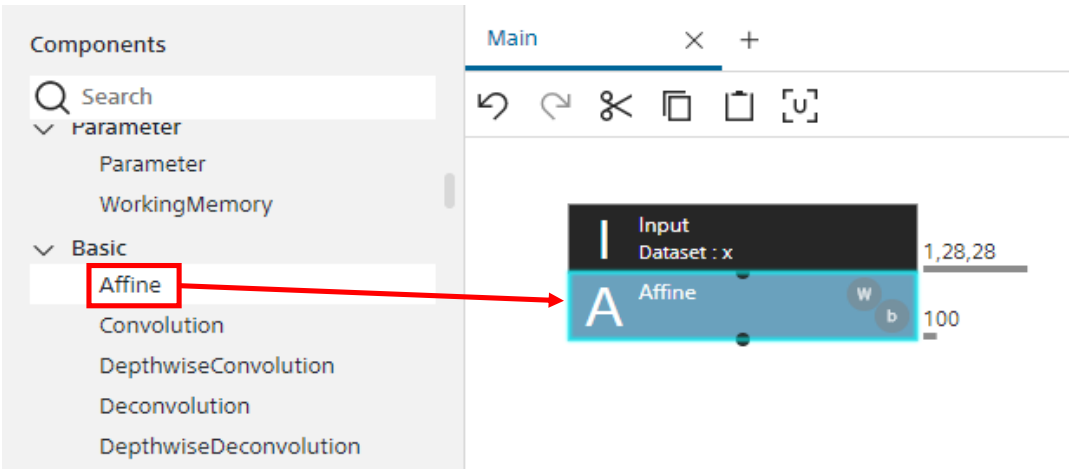


入力層を作ります。

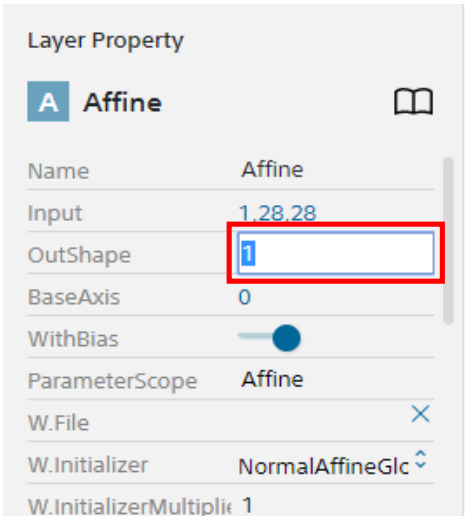
Componentsリストから「Input」を中央の領域にドラッグ＆ドロップします。



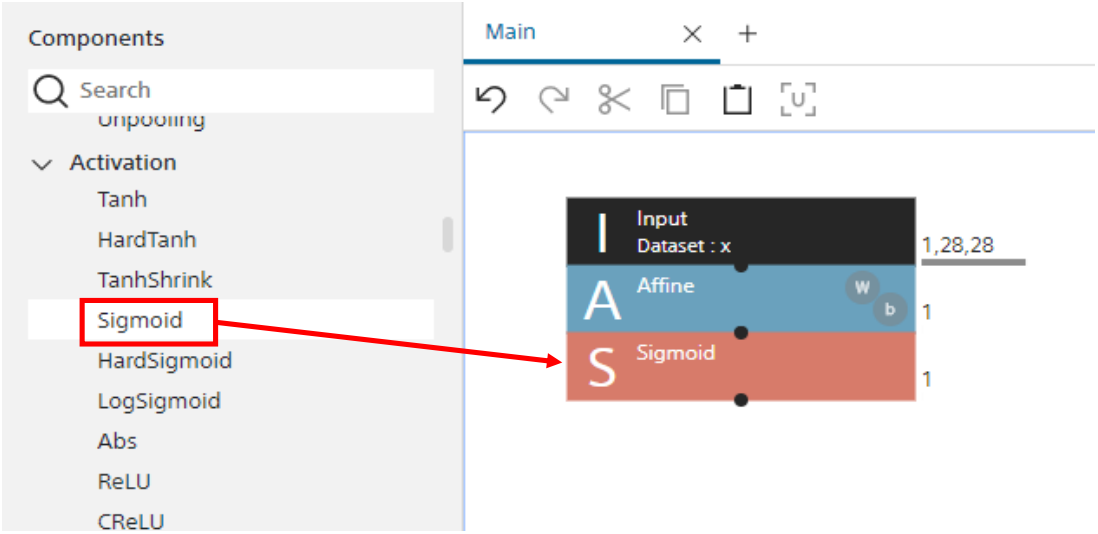
中間層の全結合層を作ります。
Componentsリストから「Affine」を先程作成したInput層の下にドラッグ＆ドロップします。



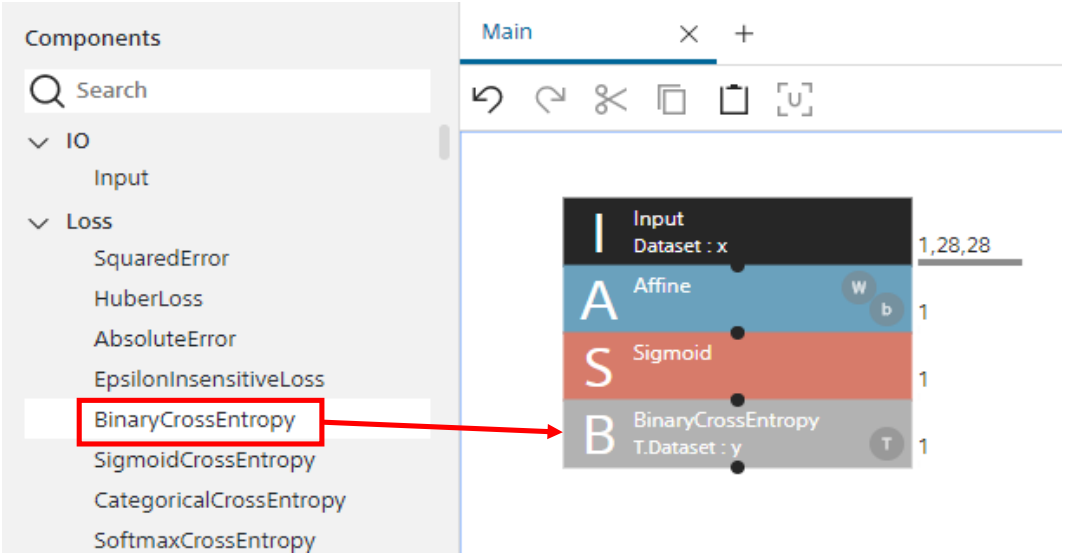
左下の Layer PropertyのOutShapeを「1」に変更します。



出力層を作ります。
Componentsリストから「Sigmoid」を先程作成したAffine層の下にドラッグ＆ドロップします。

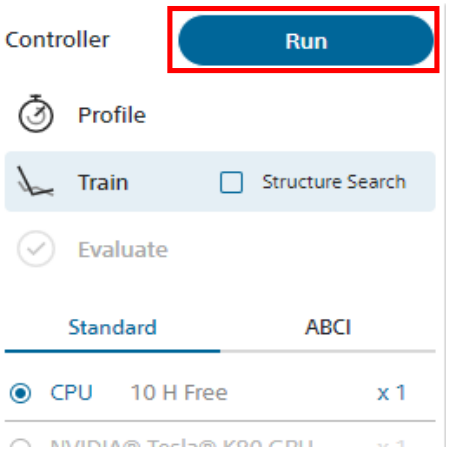


損失関数を指定します。
Componentsリストから「BinaryCrossEntropy」を先程作成したSigmoid層の下にドラッグ＆ドロップします。

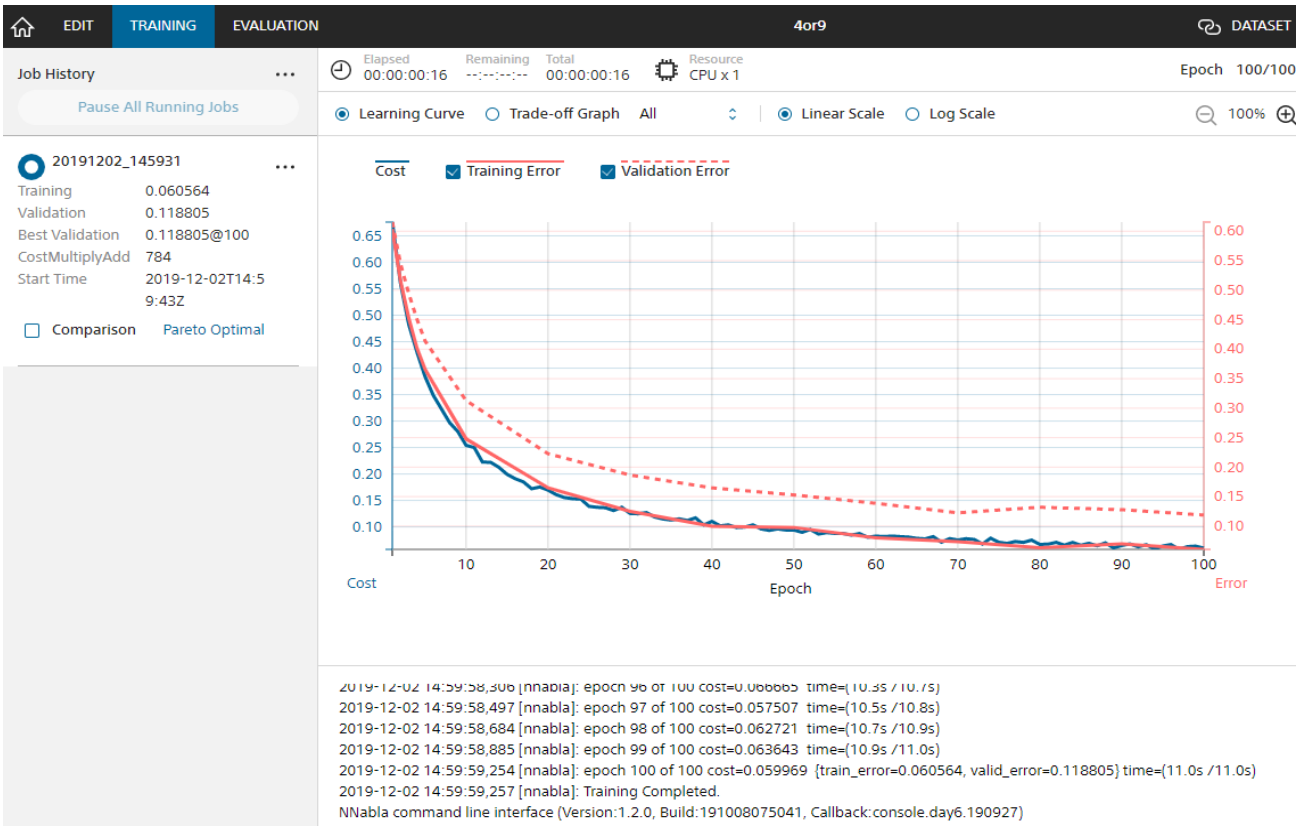


訓練の実行

右側のControllerでTrainを選択し、「Run」をクリックします。

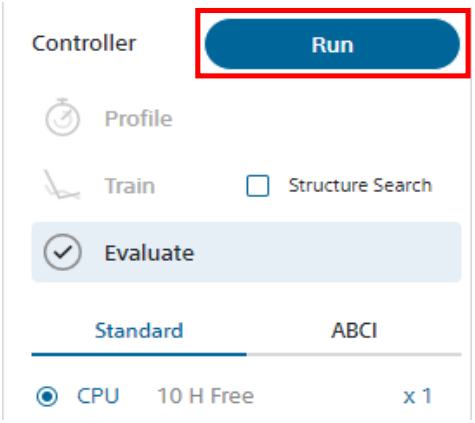


しばらくすると、TRAINING画面に切り替わり、学習曲線が表示されます。



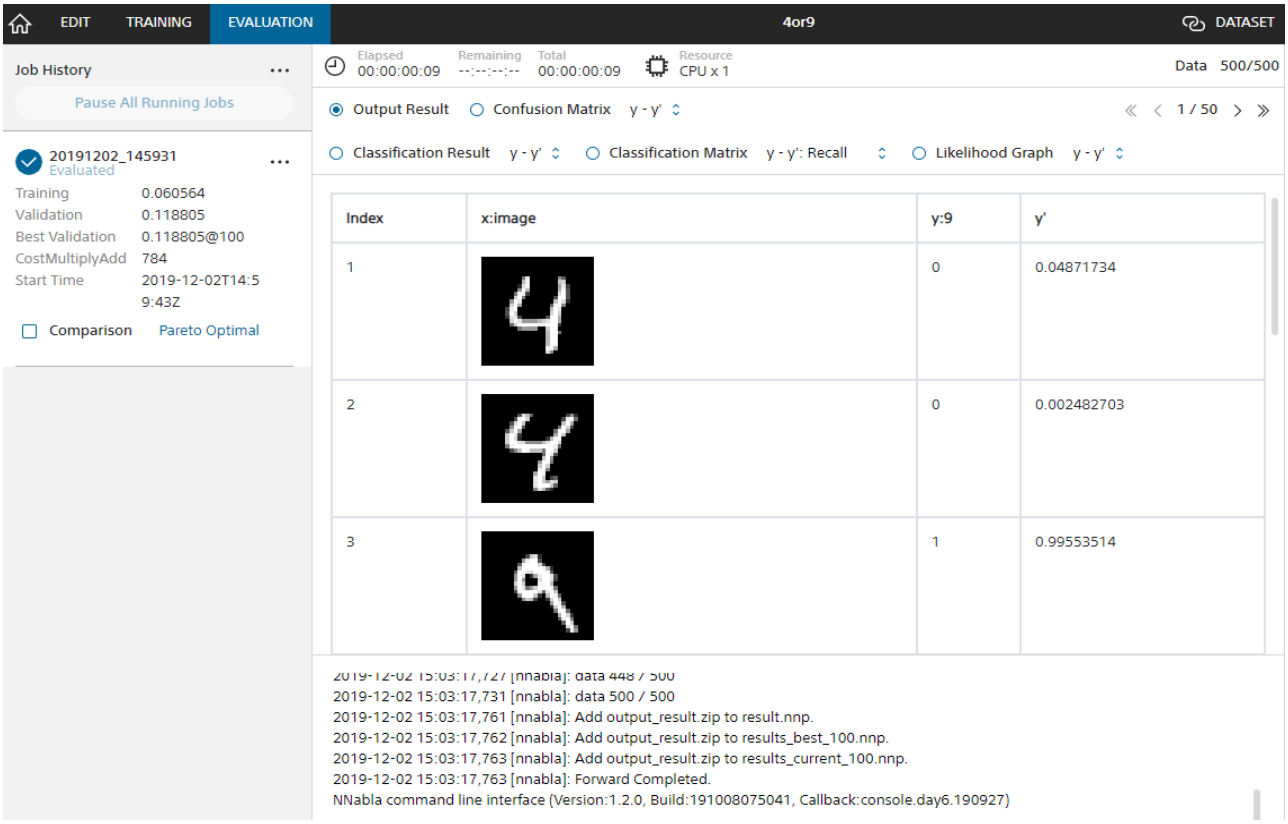
精度を評価する

右側のControllerでEvaluateを選択し、「Run」をクリックします。

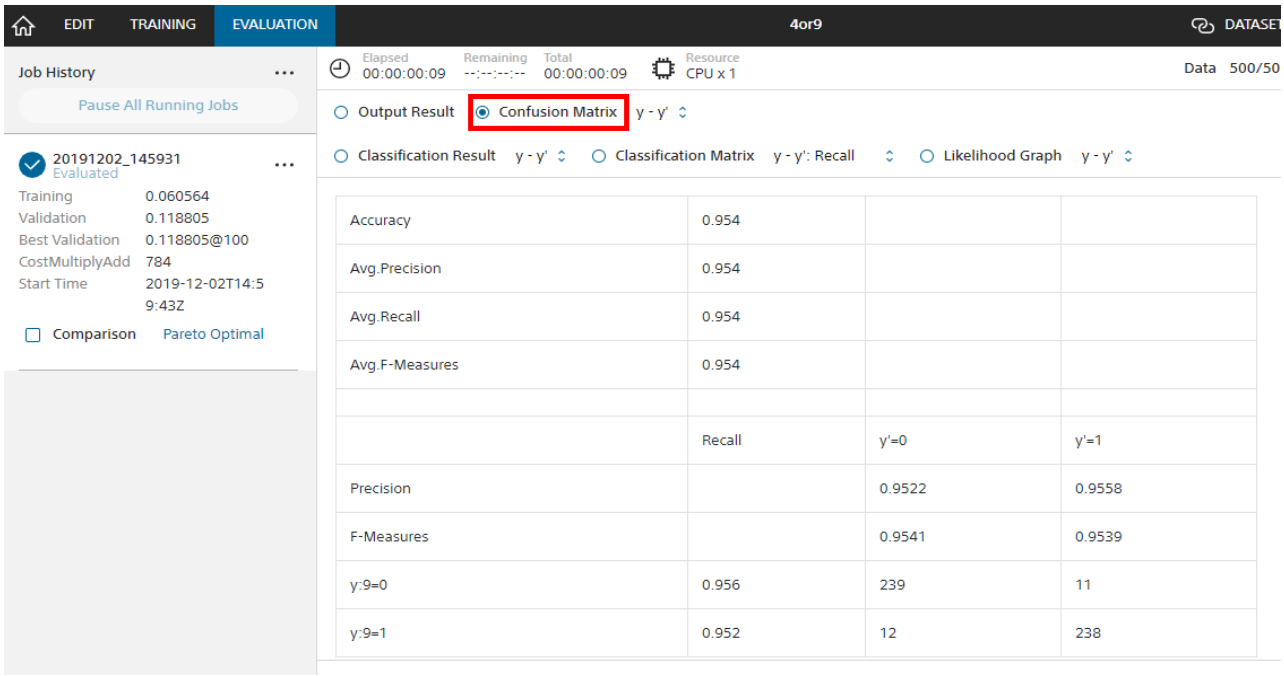


しばらくすると、EVALUATION画面に切り替わり、推論結果が表示されます。

「y:9」列に、4の画像なら「0」、9の画像なら「1」が出力されていれば正解です。



「Confusion Matrix」をクリックすると混同行列を確認できます。
Accuracy（正解率）は95.4%でした。



Accuracy	0.954		
Avg.Precision	0.954		
Avg.Recall	0.954		
Avg.F-Measures	0.954		
	Recall	y'=0	y'=1
Precision		0.9522	0.9558
F-Measures		0.9541	0.9539
y:9=0	0.956	239	11
y:9=1	0.952	12	238

演習は以上です。
モデルの構造を変更したり、他のサンプルプロジェクトを使っていろいろ試してみてください。

AI2-8

事例紹介、アイデアソン

－ 講義内容 －

- 県内動向・補助金制度紹介
- 県内ものづくり中小企業の AI・IoT 活用事例紹介
- グループワーク：アイデアソン

事例紹介

～　新潟県内　AI導入状況を調査する　～

この科目の目的

＊事例をできるだけ多く知る。

＊補助金などの制度を知る。

新潟県工業技術総合研究所
中越技術支援センター

まずは、ざっくり新潟のAI導入状況を知りたい！

2019年8月8日　ヒアリング

新潟県工業技術総合研究所様のAIへの取り組みを教えて下さい。

■ 講習会の開催
9/18（水）上越市にて、AI・IoT講演会を開催予定
http://www.iri.pref.niigata.jp/pdf/news/31new11.pdf
■ 補助金制度
■ 長岡市からの依頼で企業からの相談に対応
■ 企業との共同研究
http://www.iri.pref.niigata.jp/randd/kenr1.html

県内企業のAIへの取り組み事例を教えて下さい。

株式会社システムスクエア（長岡市）

X線を利用した異物検査機メーカー
https://www.system-square.com/

自社製品にAIを搭載。異物を検出するタスクをAIに置き換えて精度を向上させた。
・AIで魚の小骨を検出する。
（検出率：従来ソフト70%　→AI 95%）
・検査時間を従来の2割に縮める。
（検査員20秒　→　AI 4秒）
・製造過程で発生する食品ロスの低減につなげる。

関連記事（日本経済新聞）
https://www.nikkei.com/article/DGXMZO46097180U9A610C1L21000/

株式会社メビウス（新潟市）

IT・ソフトウェア業　http://www.mob.co.jp/HP/

機械学習を用いた錦鯉の個体識別システムを開発
（NICO2018年度先進技術開発支援事業）

該当URL：
http://www.mob.co.jp/HP/wp-content/themes/mob-hp/library/images/MachineLearningConsulting.pdf

株式会社アイビーシステム（新潟市）

IT・ソフトウェア業　https://www.ib-system.co.jp/

旅館内の客の行動をカメラを用いた画像解析で予測し、
従業員を効率的に配置するシステムを開発

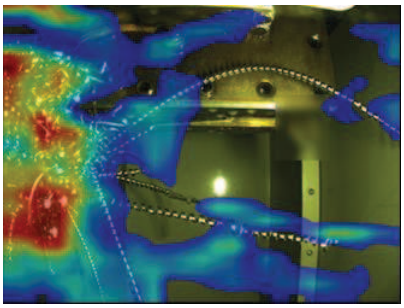
関連記事（日本経済新聞）：
https://www.nikkei.com/article/DGXMZO37184570311102018L21000/

株式会社オーエム製作所　長岡工場（長岡市）

工作機械メーカー　https://www.omltd.co.jp/

金沢工業大学と、工作機械の刃物部分への切粉巻き付き
検知の精度をAIで向上させる共同研究を行った

該当URL：
https://www.omltd.co.jp/kanri/output.php?file=news.55.1.pdf&org=②金沢工大%E3%80%80共同研究.pdf



倉敷機械株式会社（長岡市）

工作機械メーカー　http://www.kuraki.co.jp/

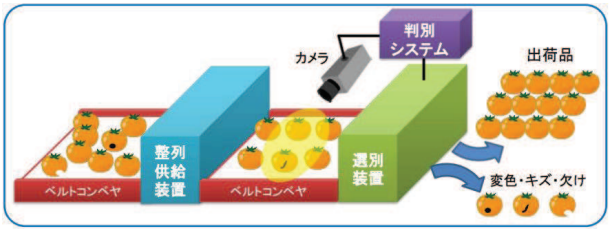
作業員の管理（今何をやっているか、無駄な作業がないか、ベテランとの違いはなにか）を
カメラで計測し画像解析を行うシステムを開発

株式会社ミツワ（燕市）

農業機械メーカー　https://kk-mitsuwa.com/

新潟県工業技術総合研究所との共同研究（平成30年度）
で、野菜色彩形状選別機を開発。
従来目視で行っていた良品・不良品、等級の判別を自動化

該当URL：
http://www.iri.pref.niigata.jp/randd/randd_pdf/h30kyodo5.pdf



新潟大学（新潟市）

教育　https://www.niigata-u.ac.jp

ル・レクチェの傷・等級を画像解析によって判別するシステムの研究
➢ 全角度からの検査に苦労した。

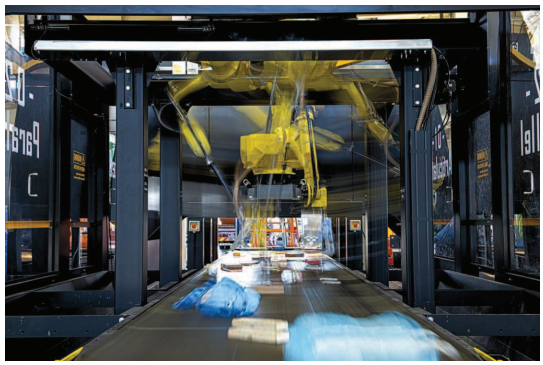
ウエノテックス株式会社（上越市）

破砕機メーカー　https://www.uenotex.co.jp/

新潟県工業技術総合研究所との共同研究（平成28・29年度）で、ゴミを素材別に分別するロボットを製品化。通常のカメラとハイパースペクトルカメラを組み合わせた画像解析により、ゴミの素材を判別

- フィンランド製廃棄物選別ロボット「ゼンロボティクスリサイクラー」の国産化を目指して作成
- ディープラーニングを用いた分別スピードは十分だが、ロボットアームの制御スピードが追いついておらず、一度に処理できる量はまだ十分ではない

該当URL：
https://www.uenotex.co.jp/blog/product1/3804



カワイ精工（上越市）

金型制作、プラスチック成形　https://www.kawai-seiko.co.jp/

- 自社WebサイトにAIによるチャットボットを実装
- AIによる生産計画の予測システムを開発、ベテランが立てる生産計画とほぼ同じ

ダイニチ工業（新潟市）

製造業　https://www.dainichi-net.co.jp/

商品の外観検査に既製のAI製品を導入

- ＡＩによる画像検査の対象は、ファンヒーターのファンを駆動するモーターのリード線が、樹脂製留め具に留まっているかを見る工程

関連記事（日刊工業新聞）：
https://www.nikkan.co.jp/articles/view/00524551

どのような企業がAIに取り組んでいますか。

長岡市では製造業が多い。補助金を受けて取り組む企業が多い。

どのような分野の製品が多いですか。

画像処理系は製品化が速い。ロボットと組み合わせるものはメカニクスの部分が難しく、製品化までこぎつけるものが少ない。

失敗事例はありますか。

- 四苦八苦しているものはたくさんある。
- 無料ツールを使うにしても、環境を整えたり、現場に持ち込むことも大変なので、セミナーなどで支援している。
- ロジックはできても、ラインのスピードに追いつけなかったり、メカの精度がネックになることもある。

行政として他に取り組んでいることはありますか。

長岡市は製造業支援に力をいれており、AIの人材育成のためのセミナーを開催している。

- 8月29日・30日にAIセミナーを開催予定
- 8月16日～18日にロボカップジャパンオープン2019を開催予定
  https://www.city.nagaoka.niigata.jp/sangyou/cate01/robocup.html

産官学連携でAI関連の取り組みはありますか。

- 長岡市に相談に来た企業の対応を、長岡技術科学大学と共同で行っている。
- 長岡技術科学大学は、国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）のAIフロンティアプログラムに採択され研究に取り組んでいる。

新潟県のAIに関する補助金

AI・IoT導入促進補助金 https://www.pref.niigata.lg.jp/sec/sangyoshinko/1356891598355.html

■ 目的
県内中小企業の生産性や付加価値の向上を図るために、AIやIoTを活用したシステム・機器等を導入して実施する、県内中小企業者のモデル的な取組を支援

■ 補助率等
補助率：補助対象経費の1/2以内、補助上限額：2,500千円/件

AI・IoT活用ビジネス創出実証業務 https://www.pref.niigata.lg.jp/sec/sangyoshinko/1356845580781.html

■ 委託内容
1. AI・IoTを活用した実証システム、ビジネスモデルの企画
   実証システムのテーマは以下のいずれかに該当するものとする。
   - 産学が連携し、AI・IoTシステムの導入により、県内複数企業の生産性や付加価値の向上の実現を図るもの
   - AI・IoTシステムで動作するロボットの活用やAI・IoTシステムにより、技術承継などの人手不足対策や人材育成に資するもの
   - 健康・医療分野におけるAI・IoT活用システムで、健康寿命の延伸や地域医療環境の向上、健康経営の推進など、「健康立県」の実現に資するもの
2. 実証システムの開発
3. 実証システムの導入・運用
4. 実証状況等の報告

■ 委託費用
1件当たり500万円以内（消費税及び地方消費税を含む。）

その他新潟のAIに関する補助金

先進技術開発支援事業費助成金（NICO） https://www.nico.or.jp/sien/hojokin/31106/

■ 目的
ロボットや高度ＩＴを活用した付加価値の高いビジネスモデルの創出及び関連産業への参 入促進に係る、次の取組みであること。

■ 助成内容
- ロボット本体、周辺機器等の試作
- 高度IT、ロボット等を活用したシステムの 試作開発
- 助成限度額：助成限度額 250万円/件、助成率：1/2以内

実証実験プロジェクト事業補助金（新潟市） https://www.niigata-iot.jp/subsidy-event/1245/

新潟市では、AI、IoTなどの先端技術を活用した民間企業等が行う実証事業への支援を行います。
生産性の向上や革新的なサービス・製品による付加価値向上などが期待できるAI、IoT、ビッグデータなどの先端技術の活用による新たなビジネスの創出を目指し、それにより新潟市における産業活力の向上、地域経済や都市の活性化を図ります。都市部や農村部など多様なフィールドを持つ本市で、新たなビジネスの創出に向けた、暮らしやすさや利便性の向上につながる意欲的な実証事業をご提案ください。

■ **実証実験補助金**
- 新規性、革新性、試験的要素の高い先端技術を活用した実証実験への支援です。
- 1件当たりの上限50万円、補助対象経費の1/2以内

■ **実証プロジェクト補助金**
- 先端技術を活用した実証を通じて、実施後の継続性や普及展開、産業活力の向上へ寄与が見込め、かつ暮らしやすさや利便性の向上につながる意欲的な実証プロジェクトへの支援です。
- 1件当たり上限1,000万円、補助対象経費の1/2以内

AI・IoT導入促進補助金

平成30年度

(株)大橋商会	IoT トラック積載量計測システムで業務改善
(株)蓬平観光	ホテル和泉屋 IoT による従業員の情報共有により業務改善を実現
(株)山口製作所	AI によるロボットピッキングで工程を自動化
(株)阿部工業	IoT による生産・出荷状況の見える化で適正な生産計画を作成
(株)伊藤建設	IoT による建設現場の測量・設計業務の効率化
第一ニットマーケティング(株)	生産性向上を目標に 100 台の横編機を IoT 化
(株)アイテック	IoT 計測器で品質記録作成を効率化

令和元年度 https://www.pref.niigata.lg.jp/sec/sangyoshinko/1356898198911.html

(有)山崎銅鉄店	資源回収時の IoT 利用による受付無人化・省力化、回収頻度と利便性の向上、およ びこれらのデータを活かすことによる営業効率・作業生産性の向上
(株)サカタ製作所	AI 画像判定を用いた組立製品の検品システム導入
(株)兼古製作所	AI ディープラーニングを用いたビット自動検査
(有)日承テクニカル	IoT 機器による作業工程のデジタル化で、稼働率、生産性の向上を図る
越後札紙(株)	IoT・AI プレス下死点圧力波形下死点精度計測システム
(株)有本電器製作所	音声入力システムを用いた生産管理システムでの情報収集方法の改善
(株)千手	IoT を活用した NFC 等物理認証による米作収穫後作業管理システム
セキサーマル(株)	IoT を活用した熱処理設備保全管理システムの導入
(有)ユー・アイ工業	IoT・AI を活用した生産・機械稼働状況監視、加工業務割振り支援システムの 導入

AI・IoT活用ビジネス創出実証業務

平成30年度 https://www.pref.niigata.lg.jp/sec/sangyoshinko/1356880153955.html

(株)山之内製作所	技能情報でつながる工場
日本電気(株)新潟支店	将来の健康リスクを予測するウェルネス予報図

令和元年度

(株)長岡計、(株)BELLSOFT	建設機械(除雪機)稼働状況管理システム
凸版印刷(株)、大日養鯉場(株)	養鯉場での ZETA 通信活用による品質管理システム

実証実験プロジェクト事業補助金（新潟市）

実証実験補助金　平成30年度 https://www.niigata-iot.jp/subsidy-event/992/

株式会社アイビーシステム	省電力長距離無線（ＬＰＷＡ）を活用した家庭用灯油タ ンク残量通知装置の実証実験事業
株式会社ＡＧＲＥＥ	医療相談アプリを用いた労働者の健康相談アクセス環 境の改善による健康行動変容促進事業（一部チャットボット）
株式会社ＮＴＴドコモ新潟支店	次世代通信規格５Ｇを活用した４Ｋ３Ｄ等高品質ＶＲ 映像伝送技術実証事業
株式会社廣瀬	スマートグラス及びソフトウェアの導入による建設現 場における遠隔作業支援事業
レントリー新潟株式会社	小型ＩＣＴ施工用建機による生産性の向上及び普及促 進事業

実証プロジェクト補助金　平成30年度 https://www.niigata-iot.jp/subsidy-event/992/

明和工業株式会社	「IoT 機器を活用したビル内の管路監視サービス」の実証事業

先進技術開発支援事業費助成金（NICO）

平成30年度

株式会社メビウス	機械学習を用いた錦鯉の個体識別システム • 養鯉業者における管理業務の効率化を目的とした、AI（機械学習）による錦鯉の個体識別システムの試作開発
株式会社ウイング	鮮魚市場内におけるスマートグラスを利用した販売業務の効率化 • スマートグラスを用いて、鮮魚市場内における販売業務をハンズフリー、ペーパーレス化するシステムの試作開発
株式会社BSNアイネット	効果的なケアプランを作成するAI支援システムの開発 • ケアマネージャーが作成するケアプランを、AIを用いて効率的かつ効果的に作成するシステムの試作開発
株式会社アイビーシステム	画像認識技術とAIを活用した館内行動予測システム • 温泉旅館内における浴場等の混雑状況を、AI・画像認識を用いて事前に予測し、通知するシステムの試作開発

令和元年度

株式会社ナンバ	自社開発のフロン漏えい検知システム『フロンキーパー』における、ＡＩプログラムを活用した漏えい警報発報システムの試作開発
株式会社アイビーシステム	fintech技術を用いたご葬儀電子決済サービスの構築
hakkai株式会社	当社固有の超精密成型技術を使った最先端ロボットの開発・量産への参入

ヒアリングの様子

イーアールエス株式会社 新潟ロボットソフトウエア開発センター

日時	2019 年 8 月 28 日(水)16:30~18:00
業種	金属工作・加工機械製造業
事業内容	ロボット及び AI による FA の展開、通信を使って IoT・ソリューションの展 開、塗工装置及び結晶体の研削・加工装置展開

■ 導入事例の紹介

三桂精機株式会社(村上市)「人手不足のためのソリューション」

- 200 種類以上の製品をもつ電設部品メーカー。人手不足により生産管理に課題を抱えていた。
- 今、何の製品をいくつ生産しているかを IoT・AI により見える化した。
  - 何の製品を作っているか(部品のサイズなど)、機械が何の動作をしているか、
  - パトランプの状態を AI でリアルタイムに画像認識。(YOLO/Caffe、フルハイ ビジョンカメラを使用。識別率 94%以上)
  - 生産数は人が目視でカウンタをメモしていたが、IoT により自動収集。
- IoT で材料切れを他部署からも監視可能にし、柔軟な人員配置を可能にした。
- 工作機械の画像認識による監視は、画像パターンが少ないため比較的容易に実現 できるとのこと。

有限会社小林製作所(新潟市)「技の継承1」

- 金属溶接・金属加工を行う。若手への技術継承に課題を抱えていた。
- データ収集のため、ヘルメットにカメラを、溶接機の電源側に電流センサを設置。
- 溶接時は手袋が外せず、手も汚れているため、文字を書くことが難しい。予め製 造番号を書いたパネルを用意し、作業の切替時はパネルをカメラに写して AI (YOLO)で文字を認識することで、何の部品を製造しているか把握できるように した。
- 収集したデータから、ベテランの溶接技能の差を示す特徴量を AI により検出し、その特徴量が映っている動画をマニュアルとして採用。また、各作業員の特徴量 を可視化することでベテランとの差を視認し、ベテランに近づけるよう訓練する システムを開発。
- 教える側の人間の生産時間を削らずに、いかにして若手への技術継承を行うかをポイントとした。

山ノ内製作所(南蒲原郡)「技の継承2」

- 航空機器メーカー。顧客の品質要求が厳しく、製品の品質向上と指導の質向上を目指している。
- データ収集のため、頭部に加速度センサ、ウェアラブルの視線検出装置、工具の位置や角度を測る 3D カメラ、工具の不可電流を測るセンサを使用。
- 作業員をベテラン・ミドル・トレーニーの 3 レベルに分け、それぞれのレベルの作業の特徴量を機械学習により抽出。抽出した特徴量を用いてディープラーニン グで技能評価・レベルの分類を行い、診断カルテと指導カルテを出力する。
- ベテランが自分でも意識していない頭部や視線の動きを可視化することで、具体的に技術を伝えることができた。
- 現在は OpenPose を用いた作業姿勢の解析にも取り組んでいる。

他社事例の紹介　～株式会社 ABEJA・武蔵精密工業株式会社 「外観異常検知」 ～

- 廉価なカメラと照明を用いて、AutoEncoder により良品の画像のみを学習に使用した不良品検知システムを開発中。識別率 97.7%。
- 外観検査は部品を全周撮影する必要がある。また、不良品と識別したものだけをラインから取り出す作業はロボットが行うことになるが、この制御が難しくコス トがかかる。

■ 成功・失敗のポイント

中小企業は最初から不良品検知にAI を導入したがるが、周辺装置の制御が煩雑であり高難易度である。まずは、簡単で効果の出やすい付帯業務・間接業務からAIを取り入れると良い。

経営層、管理層、現場が課題について共通の認識を持ち、目的・目標を設定し、AI が出す精度をどう解釈するかをあらかじめ決めておくことが重要である。

株式会社アイビーシステム

日時	2019 年 9 月 27 日(金)14:00~14:30
業種	ソフトウェア開発
事業内容	システムインテグレーション、システム開発、ロボットアプリ開発、パッケ ージソフト開発・販売
従業員数	38 名

株式会社アイビーシステムは、にいがた産業創造機構(NICO)の 2018 年度先進技術開発 支援事業に採択され、温泉旅館内における浴場等の混雑状況を AI・画像認識を用いて事前 に予測、通知するシステムの開発を行った。また、2019 年度の先進技術開発支援事業にも 採択され、精力的に AI 開発に取り組んでいる。

■ AI 開発に取り組んだ経緯

経営層が AI 開発の必要性を感じ、トップダウンで 2018 年 7 月より AI 開発を始めた。

■概要

温泉旅館「ホテル清風苑」(新発田市)と共同で、館内の混雑状況を予測して通知し、従業 員の負担減や顧客へ提供するサービスの向上につなげるためのシステムの実証実験を行っ た。

■使用した設備・技術等

- 館内の食堂や浴場前の廊下等に Web カメラを 5 台設置。 LAN は旅館既設のものを利用。
- 顧客と従業員を判別するため、従業員の教師データ画像を 100 枚使用。
- 浴場など Web カメラで直接映すことのできないエリアでは、近くの廊下を通った顧客の性別をAIで判定し、男湯・女湯のどちらに向かったかを推定。
- 性別判定モデルはオ ープンデータのデータセットを使用して学習を行った。

■ 成功したポイント

産官学連携で、長岡技術科学大学の技術協力を得られた。その繋がりで、現在では長岡技術大学内にサテライトオフィスを構えて協業を行っている。

■ 課題

館内 LAN が有線で敷設されていないエリアがあったため無線 LAN を利用したが、動画デ ータの通信負荷が高くパフォーマンスに課題が残った。

■ 開発にかかったコスト

開発費用は 500 万円。うち250 万円は先進技術開発支援事業の助成金を利用した。
プロジェクト期間は約 8 ヶ月。

■ 今後の展望

動画データの送受信は通信負荷が高いため、Google 社製 Edge TPU を利用してエッジで推論を行うことにより通信量を削減し、パフォーマンスの向上に取り組んでいる。

■ 社員教育

AI 開発に取り組んでいる社員は現在 5 名。AI 開発の開始当初に、長岡科学技術大学から技術支援として複数回講義を行ってもらった。

株式会社ユニテック

日時	2019 年 10 月 9 日(水)13:30~14:30
業種	ソフトウェア開発
事業内容	モバイル組み込み系ソフトウェアの開発、総合建設業ソリューションシステ ムの提供、実践的システムコンサルティング、アウトソーシングサービス、 ウェブコンテンツ管理システムの開発、業種別ソリューションシステムの開 発、ウェブサイトホスティングサービス、ネットワーク環境構築サービス、 IT 技術者派遣サービス、IT ソリューションに適応したハードウェアの販売
従業員数	78 名

AI 開発の概要

- 自社製品として、建設現場の作業員をサポートする**チャットボット**を開発中である。
- 建設現場において、熟練の作業員は危険な箇所を経験から判断できるが、新人には判断で きない。
- そこで、問いかけに応じてチャットボットが危険箇所や注意点などを知らせて作 業員をサポートし、安全に寄与することを目的としている。
- 株式会社植木組の協力を得て、建設現場での実用化を目指している。

■ AI 開発に取り組んだ経緯

もともと AI に関心があり、IBM ユーザー研究会のセミナーに2年間参加して、AI の知識 や IBM Cloud の開発技術を身につけていた。AI を活用したシステム開発を行いたいと考え、 ユーザが馴染みやすい チャットボットを組み込んだサービスを提案した。

■ 現在の開発フェーズ

サンプルの試作や UI の実装を経て、現在はチャットボットを作成するためのデータ収集を 行っている。

■ 使用した設備・技術等

- チャットボットの作成には IBM 社のクラウドサービス Watson Assistant を利用。
- PC・モバイル端末からの利用を想定し、Web アプリケーションとして開発している。プログラム言語は PHP。
- 応答パターンは 100 パターン程度。
- 再学習に利用するデータ収集のため、エンドユーザからフィードバックを受取る仕組 みを実装。初版リリースを早めに行い、運用の中でデータを収集して精度向上を目指 す。

■ ユーザの役割

- データの収集
- 質問テキストの教師データ作成
- 応答テキストの作成

■ 成功したポイント

事前に AI の開発技術を習得していたのが役に立った。前述の IBM ユーザー研究会の他に も、ユーザ向けセミナーなど、様々なセミナーやハンズオンに参加した。

■ 苦労した点

ユーザに教師データの作成を依頼しているが、ユーザ側担当者は AI の知識がなかったため、 教育に時間をかけた。資料を作り込み、対面で説明を重ねて理解を得た。

■ 課題

現在は PHP で実装を行っているが、Watson Assistant から PHP 用のモジュールが提供さ れておらず、Web API の呼び出し部分を自作している。保守性を考え、今後 java でリプレ ースする予定。

■ 開発期間・コスト

- プロジェクト期間は 2019 年 4 月~2020 年 3 月の 12 ヶ月。ユーザ側担当者は通常業務 の傍らで開発に携わるため、期間を長めに設定した。
- Watson Assistant の利用料金については、開発中の使用量は無料枠内に収まっている。 本稼働後はユーザと検討しながら有料プランに切り替える予定。
- Watson Assistant を利用しているため、開発コストは低く抑えられている。ここまで の開発工数は1人月程度。今後、周辺機能の作り込みも含めてトータル3人月程度の 見込み。

■ 今後の展望

- 基幹システムとの連携や法令データを参照する機能を追加予定。
  - Watson Assistant と文書解析サービスの Watson Discovery を連携させ、より細やかなサポートを可能にしたい。
- 自然言語解析以外にも、画像認識なども取り入れ機能を拡充していきたい。
  - データやモデルの販売も視野に入れている。

■ 社員教育

AI 開発に取り組んでいる社員は現在 2 名。その他に大学生のインターンシップも受け入れ、 自社内で教育を行っている。

■ 以下切り口から事例を紹介

商品開発・業界構造を変える

消費者デマンドに応える

働き方を改革する

不正・異常を検知、社会問題を解決する

■商品開発・業界構造を変える

キューピー

食品原料の異物を画像認識で検査、食の安全守るため装置は同業に外販も
https://tech.nikkeibp.co.jp/atcl/nxt/mag/nc/18/051600049/051600001/

- 対象
✓ ジャガイモをサイコロ状にカットしたダイスポテト

- 特徴
✓ 不良品のデータを教師データとして学習させて不良品を検出する
✓ 不良品の登録は現実的でない
✓ 良品を学習させて、良品以外はすべて不良品とする

- 課題
✓ 現場運用するため安全性の確認、工場でのモノや情報の流れを整流化する対応
✓ 最適な照明やカメラの撮影条件などフィジカル系のチューニング
✓ ベルトコンベアーの認識スピード、エッジの汎用的コンピューターの検討

- 成果
✓ キューピー３工場で稼働中、不良品の検出率は100%を達成
✓ 良品を不良品として判断するのは1万個に1個程度

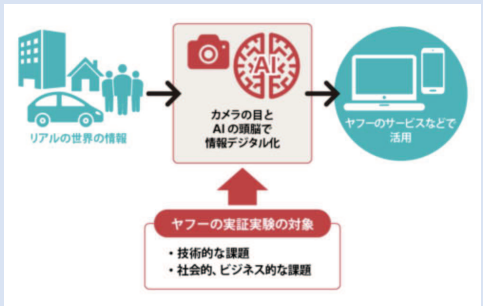
■消費者デマンドに応える

ヤフー

安いガソリン、空いている駐車場を近くで探したい、ディープラーニングとドラレコで
https://about.yahoo.co.jp/pr/release/2018/10/15a/

- 対象
✓ リアルな情報をディープラーニングでデジタル化し、今後のサービス向上につなげる。
✓ 第一歩（実証実験）として駐車場の満空情報、ガソリンスタンドの価格情報

- 特徴
✓ アスクルの配送者130台にカメラを搭載して情報を集める。
  - 予め地図情報で登録した地点のデータのみサーバに送る工夫
  - 個人情報（顔やナンバー）はサーバに送った時点で消去



- 課題
✓ ドライブレコーダーのCPUでモデルを実行
  - ドライブレコーダーの排熱が難しく、短時間で熱で停止
✓ １日に２回程度のデータ取得ではリアルタイム情報にならない
  - ドラレコなど既に搭載している企業との協業の必要性

- 成果
✓ コインパーキング３か月程度で70%認識率
✓ ガソリンスタンド情報は油種なし70%、油種含んで60%

授業の内容

1　AI導入事例を紹介し、業務課題に対しAI導入の背景やメリットを理解する

2　AI導入事例について、AI導入に際して候補となるAI技術要素を理解する

3　グループにて、業務課題の解決方法として、AI導入の可能性を話し合う(ワークショップ)

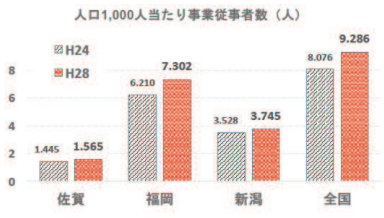
4　アンケートをとおして、講座全体の振り返りを行う

トピックス①

佐賀のIT産業は、今…

佐賀県産業スマート化センター
Industry4.0 AI-IoT Business Innovation

過去4年間で労働生産性が改善、ほぼ福岡並みに



従事者数はやや増、だが未だ少ない

労働生産性は大幅改善、今や福岡並み

出所）経済センサス活動調査（H24、H28）…中分類の「情報サービス業」及び「インターネット付随サービス業」の合計

県内のIT企業等の動向

AIやIoT、RPA等によるビジネス創出	木村情報、オプティム、福博印刷、佐賀電算、ITインペルなど
ARやVR、MRなどのコンテンツ技術	とっぺん、ウェアサーブ、神風プロダクション、九州コーユーなど
シェアリングエコノミーなど	価値創造プラットフォーム、manaview、クラウドファンディング総研など

佐賀県におけるAIやIoTといった先進技術の 利活用促進に関する取組について
https://www.pref.niigata.lg.jp/uploaded/attachment/182503.pdf

トピックス② AI戦略2019(内閣府)

社会実装に関する主な取組

地球規模課題及び我が国の課題を克服し、多様性を内包した持続可能な社会を実現するため、
我が国の強い技術とAIを融合して、価値創造と生産性向上、産業競争力を強化

システム・アーキテクチャの設計・構築

- 米国NIST等を参考に、国全体の研究開発成果の社会実装を促すための**システム・アーキテクチャを設計・構築**
- まずは**重点５分野**において、アーキテクチャ設計に基づくデータ基盤を踏まえた社会実装を**世界に先駆けて実現**
- アーキテクチャ設計を行う**専門家による体制を構築**、加えて米国NISTやドイツの**関係機関との連携を検討**

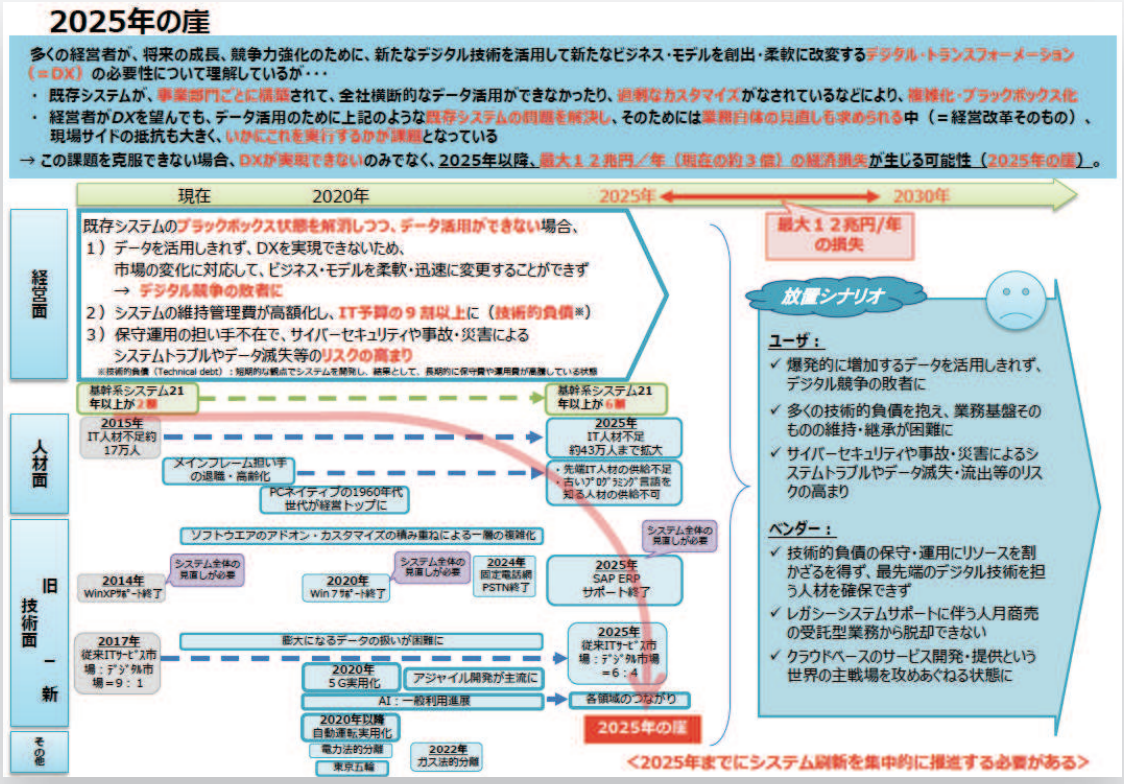
①健康・医療・介護	②農業	③国土強靭化（インフラ・防災）	④交通インフラ・物流
データ基盤の整備	スマート農業技術の現場導入	インフラ業務における新技術等の開発・導入	人的要因による事故のゼロ化
日本が強い分野（画像診断等）のAI技術開発	スマート農業の実現による、農業の成長産業化	インフラデータプラットフォームの構築	移動に伴う社会コストの最小化
予防・介護へのAI導入	農業分野におけるAI人材の育成	AIを活用した強靭なまちづくり	物流網における生産性向上・高付加価値化
世界最先端の医療AIハブ			
医療従事者リカレント教育			

⑤地方創生（スマートシティ）

日本発のスマートシティを再定義し、その実現に向けた**インクルージョン・テクノロジー**の開発と、**スマートシティプラットフォーム**の形成

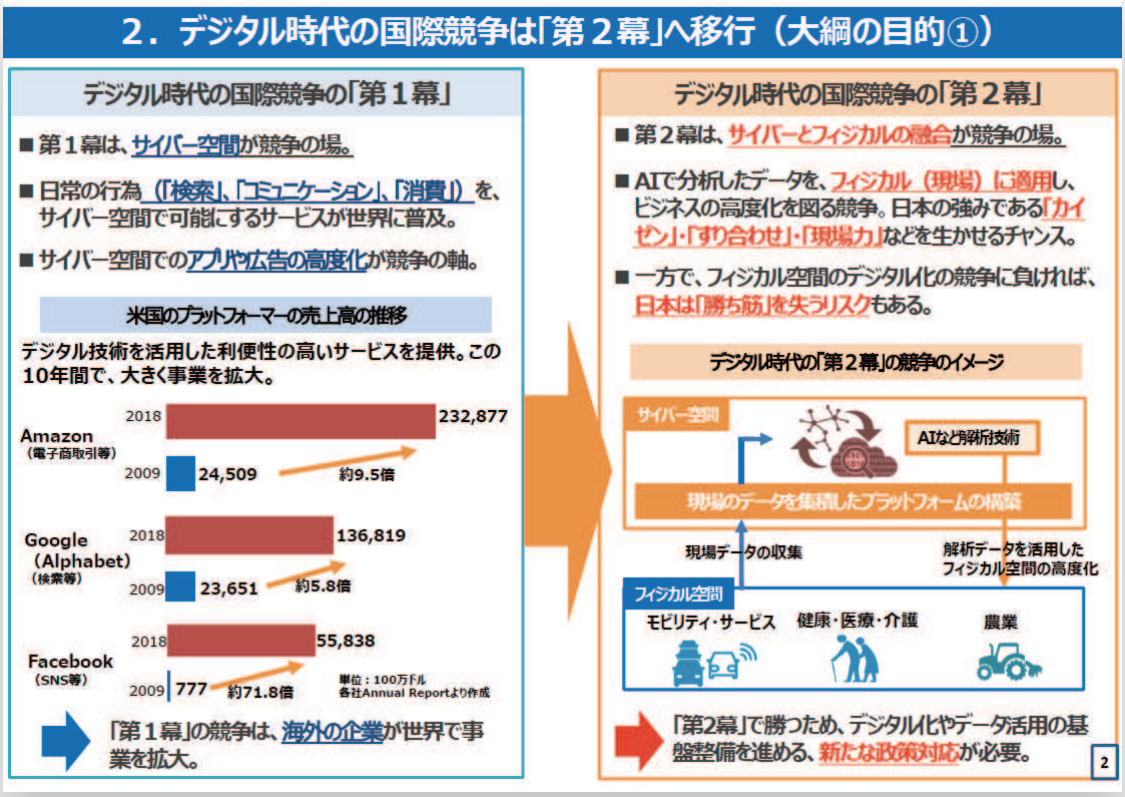
AI戦略 2019～人・産業・地域・政府全てにAI～（統合イノベーション戦略推進会議決定）令和元年6月11日
https://www.kantei.go.jp/jp/singi/tougou-innovation/pdf/aisenryaku2019.pdf

トピックス③ ITシステム「2025年の崖」克服とDXの本格的な展開（経産省）



DXレポート ～ITシステム「2025年の崖」克服とDXの本格的な展開～（経産省 平成30年9月7日）
https://www.meti.go.jp/shingikai/mono_info_service/digital_transformation/20180907_report.html

トピックス④ デジタル時代の新たなIT政策大綱（首相官邸）



デジタル時代の新たなIT政策大綱(首相官邸) 令和元年6月7日
https://cio.go.jp/node/2534

AI活用の取組み①

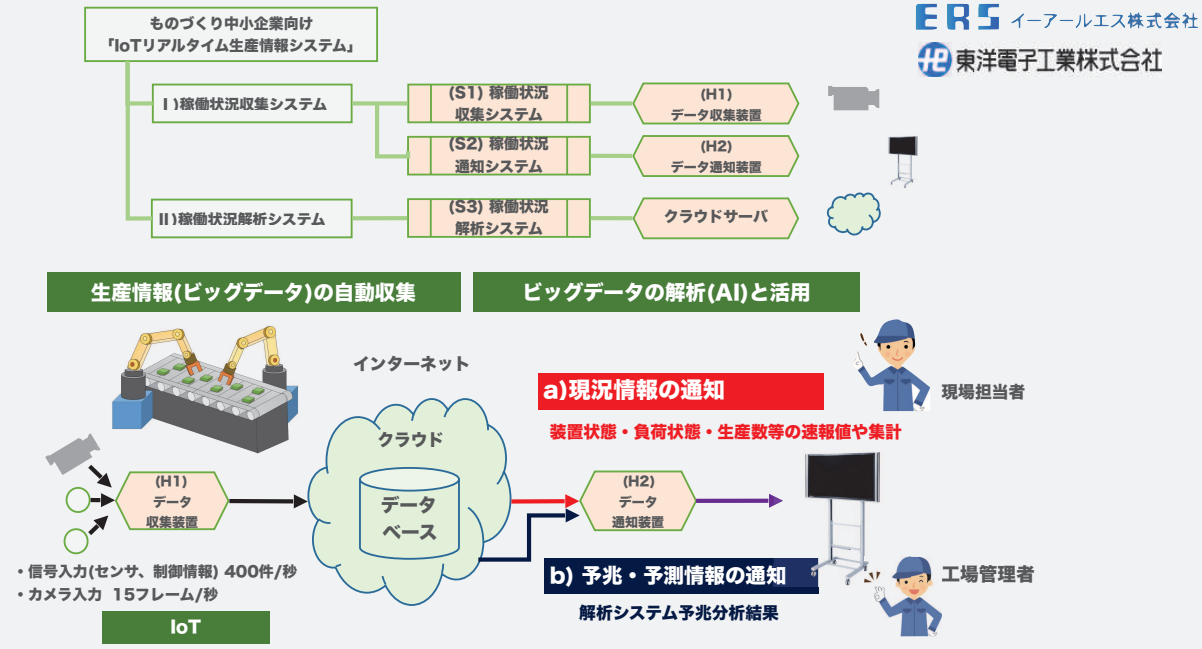
県内ものづくり中小企業が抱える問題点

- 人手不足
- 技能継承
- 事業承継

① 人手不足を解決するためのAI・IoT活用

課題 — 方策 — 目標 — **導入** → 効果

新潟県AI・IoTビジネス創出実証 ものづくり中小企業向け「IoTリアルタイム生産情報システム」



ものづくり中小企業向け「IoTリアルタイム生産情報システム」
https://www.pref.niigata.lg.jp/uploaded/attachment/82706.pdf

AI活用の取組み②

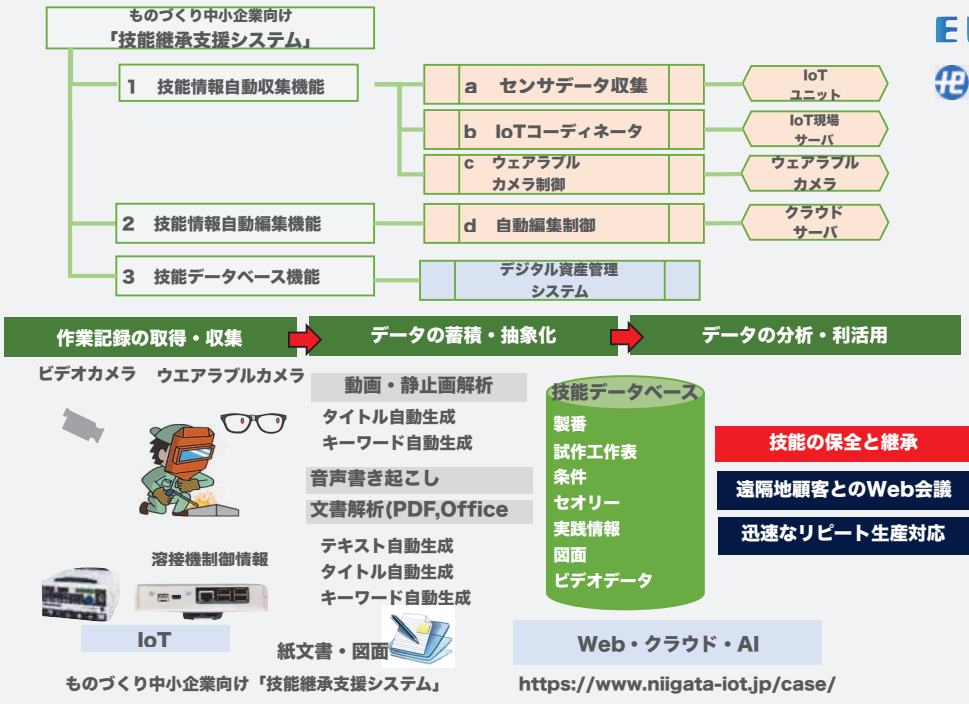
県内ものづくり中小企業が抱える問題点

- 人手不足
- **技能継承**
- 事業承継

② 技能継承を支援するためのAI・IoT活用

課題 ― 方策 ― 目標 ― **導入** → 効果

新潟市製造業IoTチャレンジ事業　ものづくり中小企業向け「技能継承支援システム」



ERS イーアールエス株式会社

東洋電子工業株式会社

ものづくり中小企業向け「技能継承支援システム」　https://www.niigata-iot.jp/case/

閑話　監視カメラの人物検知を回避する2D画像

掲載日時： 2019/05/01 09:00:00

cnet Japan 監視カメラの人物検知を回避する2D画像--ベルギーの大学が開発

ベルギーのルーヴェン・カトリック大学（KU Leuven）の学者らは、シャツやバッグに印刷できるシンプルな2D画像を開発した。この画像を持ってカメラの前に立つと、機械学習を用いている監視カメラシステムで検知されなくなるという。

著者：翻訳校正： （Special to ZDNet.com） 編集部 Catalin Cimpanu

URL：https://japan.cnet.com/article/35136333/

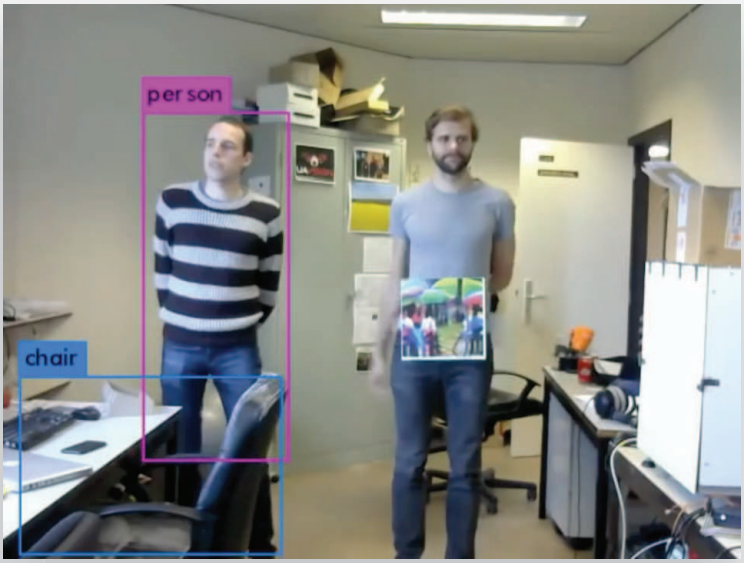
ベルギーのルーヴェン・カトリック大学（KU Leuven）の研究者らが、監視カメラの人物検知を免れる方法を考案した。シャツやバッグに印刷できるシンプルな2D画像を持ってカメラの前に立つと、ライブ動画フィードの人間を認識するのに機械学習を用いている監視カメラシステムで検知されなくなる。

patchを持っていない人（左）はシステムに検出されるが、patchをもった人（右）は検出されない
提供：KU Leuven

研究論文で「patch」と呼ばれているこの2D画像が監視カメラを欺くには、画像が人間用「検知フレーム」の中央付近に配置され、常に監視カメラに向いていることが条件となる。

この画像は動画フィードから人間の本来の顔を隠すわけではないが（その人物がマスクをしている場合は別だが）、人間検知アルゴリズムが人体を検知しなくなり、その後の顔認識チェックも起

監視カメラの人物検知を回避する2D画像--ベルギーの大学が開発
https://japan.cnet.com/article/35136333/



Generating adversarial patches against YOLOv2
https://www.youtube.com/watch?v=MIbFvK2S9g8

休題　レーザー照射でスマートスピーカーをハッキング

2019/11/30　レーザー照射でスマートスピーカーをハッキング可能、研究者らが指摘 - CNET Japan

掲載日時： 2019/11/06 10:32:00

cnet Japan レーザー照射でスマートスピーカーをハッキング可能、研究者らが指摘

レーザーによってグーグルやアマゾンのスマートスピーカーを遠隔操作できるという、予想されていなかった脆弱性が研究者らによって発見された。

著者： （CNET News） 翻訳校正： 編集部 SEAN KEANE

URL：https://japan.cnet.com/article/35144955/

スマートスピーカーについてはかなり以前から、プライバシーの問題とハッキングの懸念が指摘されてきたが、レーザーによって遠隔操作が可能という、予想されていなかった脆弱性が研究者らによって発見された。電気通信大学とミシガン大学のチームは、レーザー光線をマイクに照射することにより、Amazonの「Alexa」、「Googleアシスタント」、Appleの「Siri」を操作できることを発表した。

Amazonの「Echo Dot」
提供：Ben Fox Rubin/CNET

米国時間11月4日に公開され、The New York Times（NYT）によって報じられた同チームの論文には、230～350フィート（約70～107m）離れた場所から望遠レンズでレーザーを照射することにより、「Google Home」にガレージのドアを開けさせることができたと記されている。

https://japan.cnet.com/print/35144955/　1/2

レーザー照射でスマートスピーカーをハッキング可能、研究者らが指摘
https://japan.cnet.com/article/35144955/

Light Commands Demo #3 - Through a Window
https://www.youtube.com/watch?v=EtzP-mCwNAs

AI活用の取組み③

県内ものづくり中小企業が抱える問題点

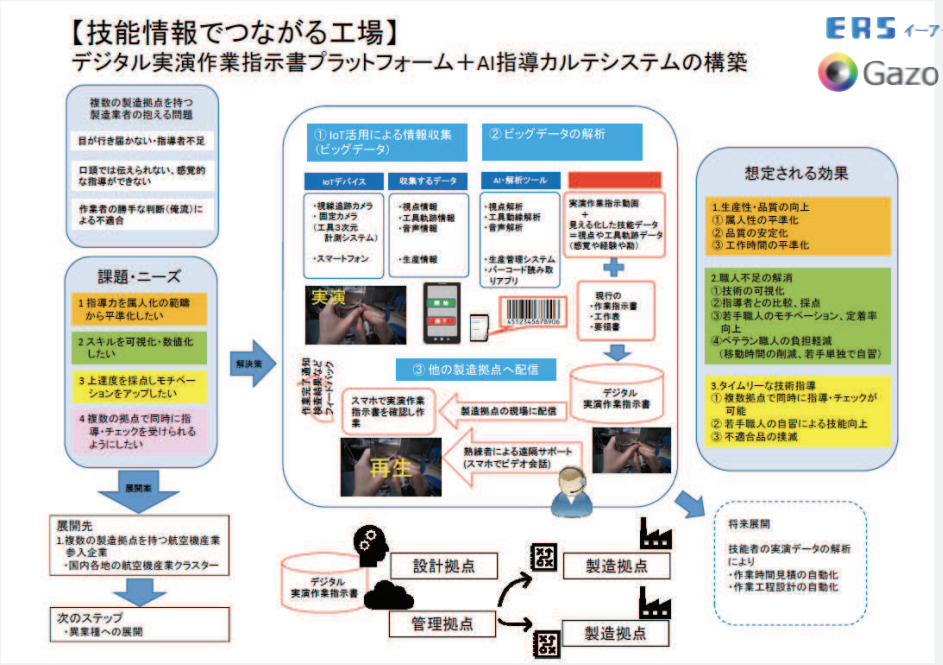
人手不足

技能継承②

事業承継

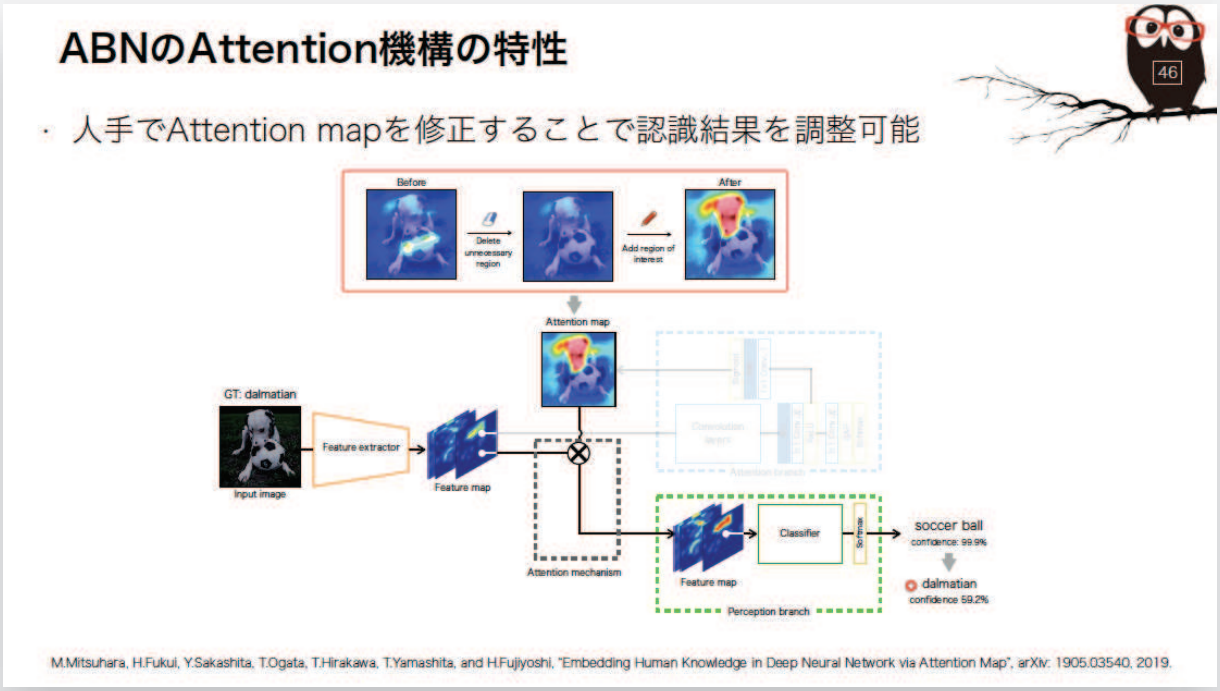
② 技能継承を支援するためのAI・IoT活用

課題 — 方策 — 目標 — **導入** → 効果



「技能情報でつながる工場」デジタル実演作業指示書プラットフォーム＋AI指導カルテシステムの構築
https://www.pref.niigata.lg.jp/uploaded/attachment/82700.pdf

閑話 Attention Branch Network (注目箇所可視化技術)の応用



最近の深層学習におけるAttention機構

https://speakerdeck.com/fhiro/zui-jin-falseshen-ceng-xue-xi-niokeruattentionji-gou-ming-gu-wu-cvprmlmian-qiang-hui-ver

中部大学 機械知覚ロボティクスグループ http://mprg.jp/research/abn_j

PyTorchによる実装@git https://github.com/machine-perception-robotics-group/attention_branch_network

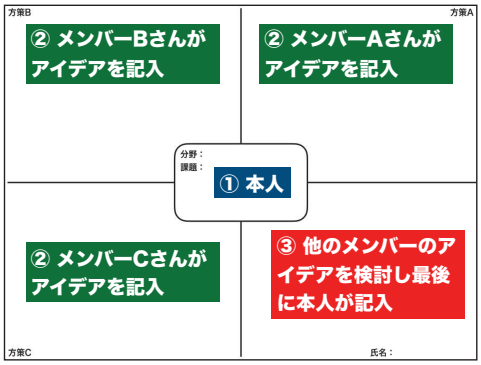
アイデアソン　目的

- 協働して共創する練習
- 短時間の検討を繰返してスパイラルアップする練習
- 成果物の完成度は求めない。思考過程が重要
- 協働相手とのコミュニケーション、検討の多様性や多角性、柔軟性が重要
- 班内２名間で共創→班内全員で共創→講座全員の順で共創しAIビジネス企画書を作成する

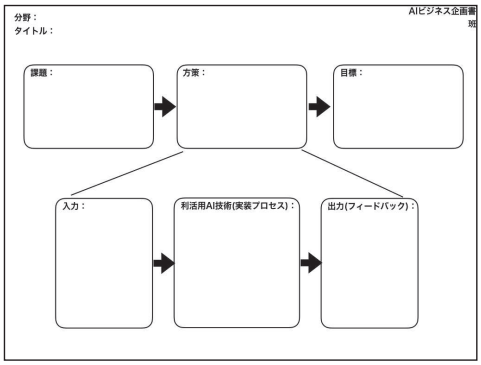
アイデアソン　手順

① 各自が課題検討シートの中央に、自社や顧客の課題と分野を記入する	14:45～14:50 5分
② 課題検討シートを班内で回覧し、他のメンバーがその課題解決のアイデアを外側の欄に記入する	14:50～15:05 5分×3 15分
③ 本人が他のメンバーのアイデア参考にして本人のアイデアをまとめて右下欄に記入する	15:05～15:10 5分
④ 班内で各自1分程度でプレゼンを行い全員で検討。事業素質の高いものを1件選定し、班内でAIビジネス企画書にまとめる	15:10～15:20 10分
	(休憩10分)
⑤ 各班ごとに代表者が5分程度でAIビジネス企画書のプレゼンを行い、他の班から一人ずつ質疑応答・コメントする	15:30～15:40 15:50～16:00 16:00～16:10 16:10～16:20 10分×4 40分
⑥ 最良の企画書を選定する。	16:20～16:25 5分

※時間配分は目安



課題検討シート
各自１枚



AIビジネス企画書
各班１枚

アイデアソン　課題の分野（事前設定）

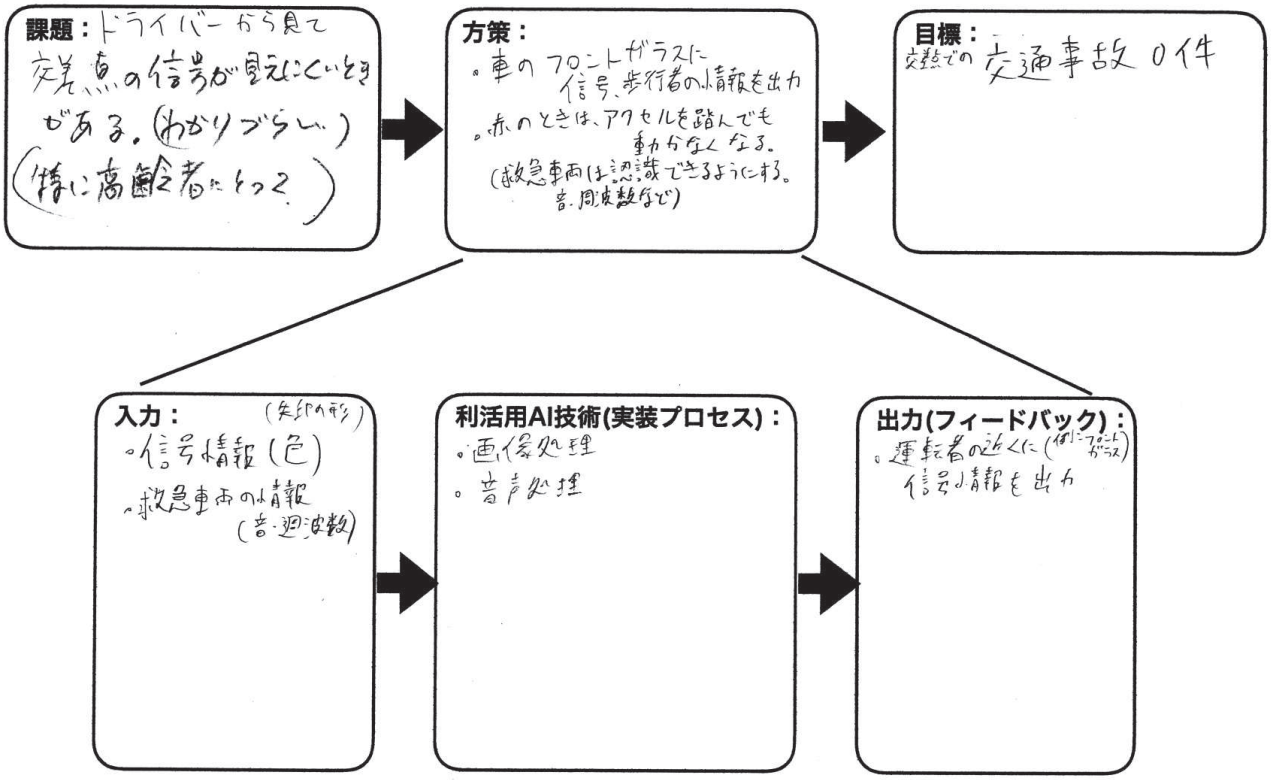
①	農業	（生産性向上、高機能化など）
②	健康・医療・介護	（サービス向上、予防、介護など）
③	交通・物流	（生産性向上、ナビサービスなど）
④	国土強靭化	（環境、防災、災害対応など）
⑤	製造業	（生産性向上、高付加価値化、SCMなど）
⑥	流通サービス	（マーケティング、高付加価値化、受発注など）
⑦	その他	（観光、スポーツ、エンタメなど）

アイデアソン　成果物

ビジネス企画書

分野： 交通・物流
タイトル： 高齢者の交通事故撲滅

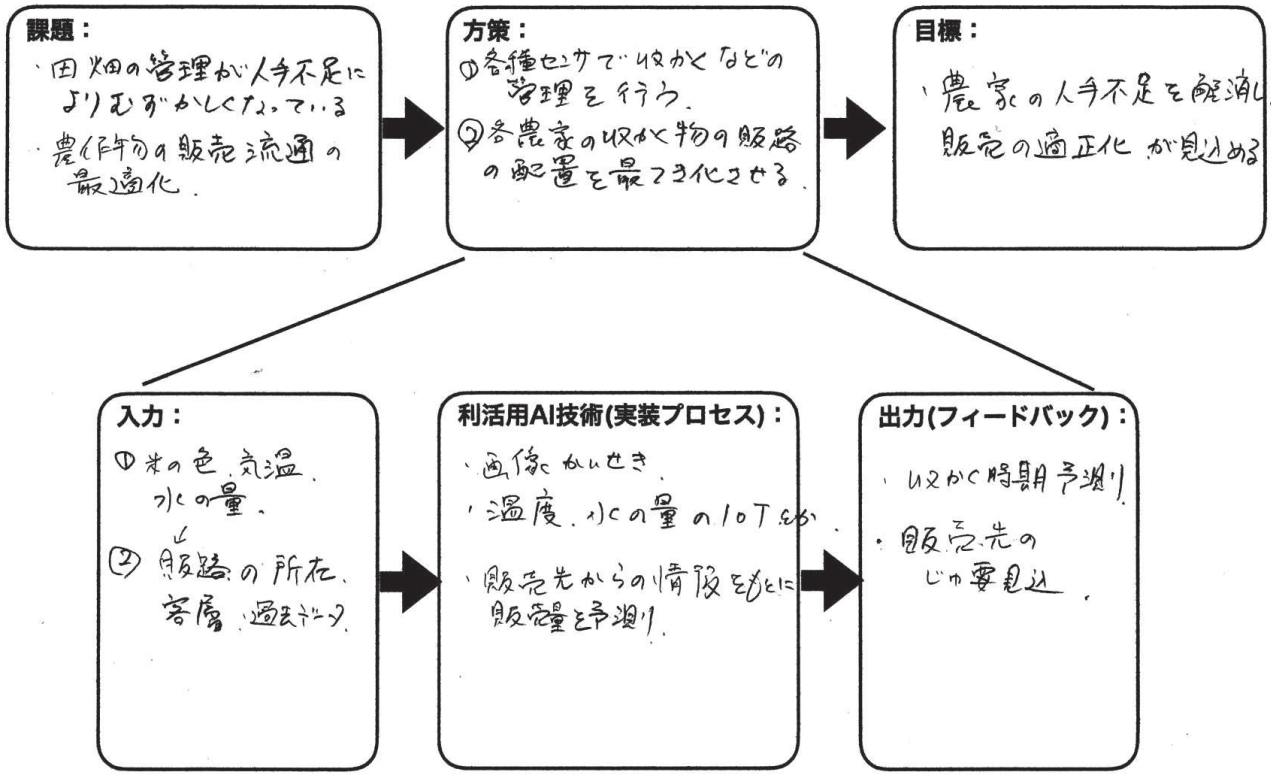
AIビジネス企画書
A 班



分野： 農業

タイトル： 農家の生産性・収入向上

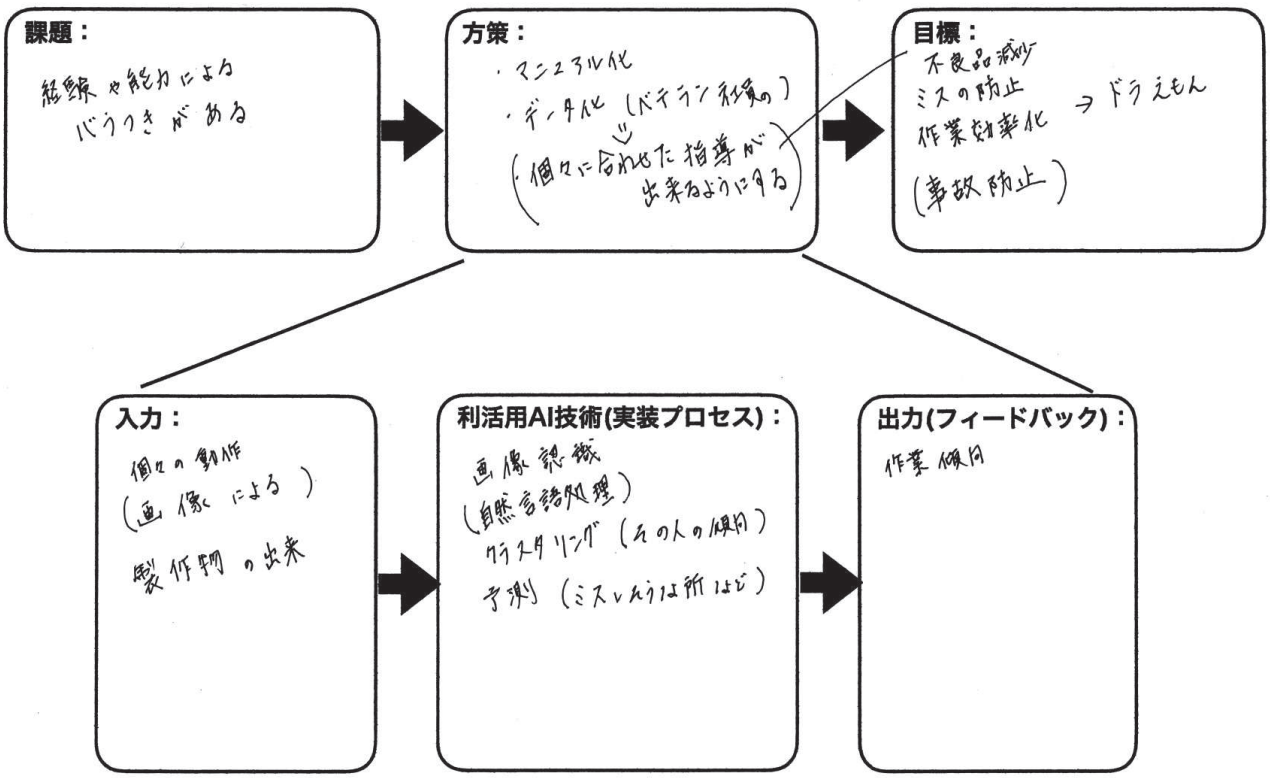
AIビジネス企画書
B 班



分野： 製造

タイトル： 作業お助けAI

AIビジネス企画書
C 班



分野： 運送（宅配）
タイトル：

AIビジネス企画書
D 班

課題：

荷物届けの隙の不在解消
（特に学生）

方策：

①受取人のデータ取得・分析
・配送ルートから時間提示

目標：

・確実に荷物を
受け取ること

入力：

①過去の受取実績
・曜日
・時間
・職業
・電力量
・家族構成

利活用AI技術(実装プロセス)：

・回帰予測

出力(フィードバック)：

在宅率
↓
最適配送ルート
の選出