第 3 章

機械学習（教師あり学習）

教師あり学習にはどのような手法があって、どんな課題解決に役立てられているのでしょうか。この章では、実践例とともに、手法の種類とそのアルゴリズムについて学んでいきます。

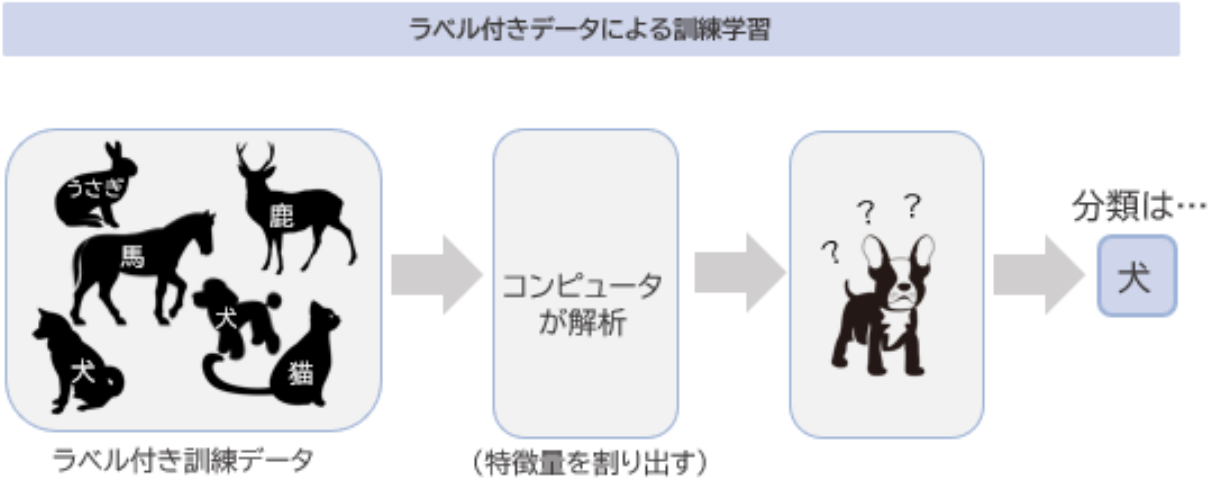
この章のゴール

教師あり学習の基本的な手法と実践例を理解する

教師あり学習とは

教師あり学習とは、既知となった過去の入力データと出力データを機械学習アルゴリズムにあらかじめ与えることで、それらを「正解データ」として計算する機械学習の手法です。

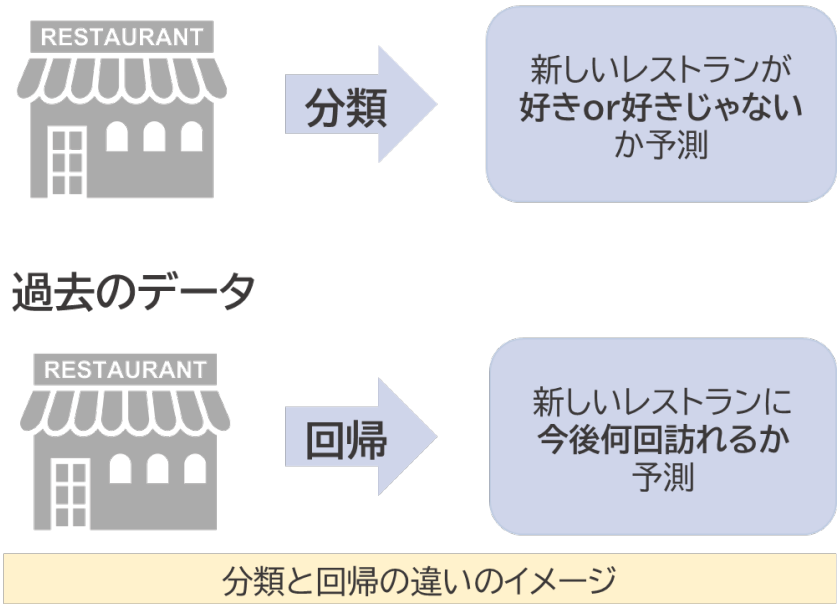
例えば、大量の動物の画像データが存在した時に、「これは“ネコ”」「これは“イヌ”」・・・といったようにあらかじめラベリングをしておきます。十分な正解データを用意し、それらを教師（正解）として機械学習を行います。未学習の画像を読み込ませた場合にも、正解の中から一致するデータを見つけ出し、“ネコ”か“イヌ”を判定することができます。



＜身近な「教師あり学習」の活用例＞
私たちの身近の以下のような場面で、教師あり学習は活用することが可能です。

- 受信した電子メールがスパム（迷惑メール）かどうかを自動判定する
- 新築の住宅販売価格を、過去の実績に基づいて予測する
- 選挙において、ある候補者についての投票率を予測する
- 住宅ローン申込者への融資リスクの高低を判断する

教師あり学習は、あらかじめ定めた分類に学習データに振り分けるための「分類」と、連続するデータの将来の値を予測する「回帰」の２つのタスクに分かれます。



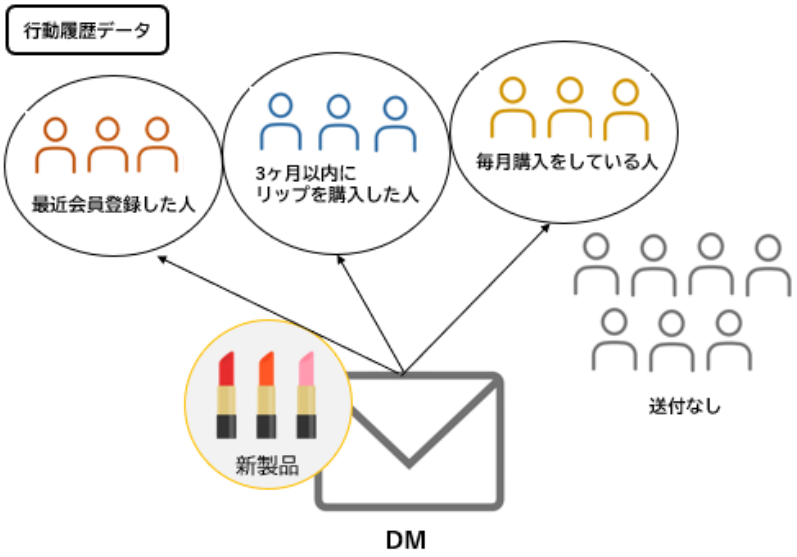
分類と回帰の違いのイメージ

1. 分類

分類の主な目的は、分析したいデータが所属するクラス分けを予測することです。特に、YES or NO のように予測対象のクラス数が 2 つの場合、**二値分類**と呼ばれます。

＜例＞

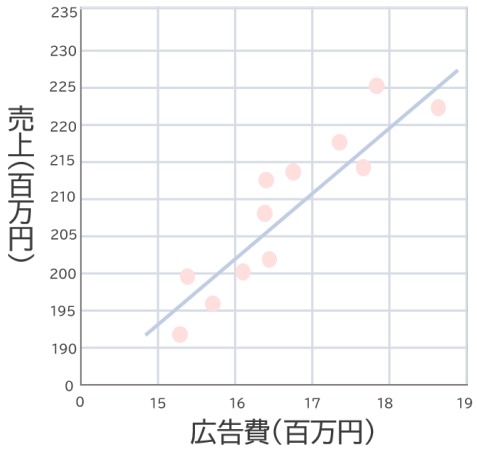
化粧品ブランド A で新商品を発売することになりました。顧客向けに新商品の案内を行う時に、過去の購買履歴データを使って、なるべく新商品を買ってくれそうな人を予測して、その人たちのみに DM を送付します。



2. 回帰

回帰の主な目的は、連続する値の傾向をもとに予測を行うことです。

具体的な活用例では、企業が商品やサービスの広告費用の増額を検討する際に、「広告費を増やすことでどのくらいの売上を見込めるか？」などの予測する際に使われます。



回帰分析は、結果となる数値と要因となる数値の関係を調べ、それぞれの関係を明らかにする統計的手法です。このとき、要因となる数値を「説明変数」、結果となる数値を「目的変数」といい、「説明変数」が１つの場合を「単回帰分析」、複数の場合を「重回帰分析」といいます。

事例：重回帰分析を活用して、飲食店の売上予測を行う

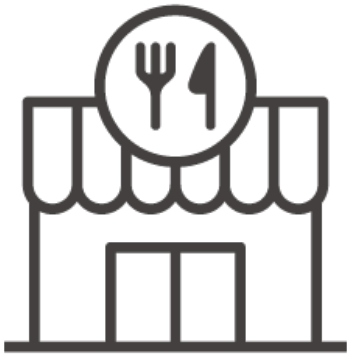
飲食店業界などでは、将来的な店舗の売上がどのくらいになるかをあらかじめ予測するために重回帰分析をベースとした機械学習の手法を用いられることが多いです。店舗の来客数やオーダーされた商品の履歴など、様々なデータを活用して売上予測金額を算出します。

では実際に、このような売上予測（需要予測ともいいます）ではどのようなデータを用いることができるのかを考えてみましょう。

売上に影響を与える変数にはどのようなものがあるでしょうか。このとき、求めたい値は「売上金額」でこれが「目的変数」となります。この目的変数は、他のどのデータで決まるのでしょうか？ もし仮に、曜日、近隣でのイベント開催有無、店舗からのチラシ配布、天気などが影響していたら、それらを「説明変数」として予測することになります。

	最高気温（℃）	最高気温（℃）	天気
2020/01/01	8.4	-1.3	晴れ
2020/01/02	11.4	-1.5	晴れ
2020/01/03	10.0	2.0	曇り
2020/01/04	6.8	0.5	晴れ
2020/01/05	12.7	1.2	曇り
2020/01/06	6.5	-2.3	雨
2020/01/07	8.4	0.2	雨
2020/01/08	7.8	1.0	曇り
2020/01/09	10.5	2.3	晴れ
2020/01/10	10.8	2.0	晴れ

	広告チラシ配布有無	イベント開催
2020/01/01	なし	なし
2020/01/02	なし	あり
2020/01/03	あり	あり
2020/01/04	あり	あり
2020/01/05	あり	なし
2020/01/06	なし	なし
2020/01/07	あり	あり
2020/01/08	なし	あり
2020/01/09	なし	なし
2020/01/10	あり	なし



予測モデルおよび予測結果の評価（教師データと評価データに分割して検証する交差検証法）

教師あり学習では、収集したデータを教師データ（学習データ）と評価データの 2 つに分けて学習を行います。このようにデータを分割して評価することを「交差検証法（クロスバリデーション）」と呼びます。

教師あり学習は、人間が付けた正解のラベルに基づいて機械が学習を行った後に別のデータでの予測を行うモデルを構築するものですが、「正解のラベル」を付けてモデルを作成する工程を教師データで行い、予測は評価データを用いて行われます。「予測した結果がどのくらい合っているのか」が、モデルの精度となります。教師データ全てを用いてモデルの構築を行うと、そのデータには適合することができても、その後入ってくる未知のデータには全く合わないモデルが形成されてしまうことがあります。これを過学習（オーバーフィッティング）といいます。この過学習を防ぐために、手元にあるデータを教師データと評価データに分割してモデルの構築と予測を行います。

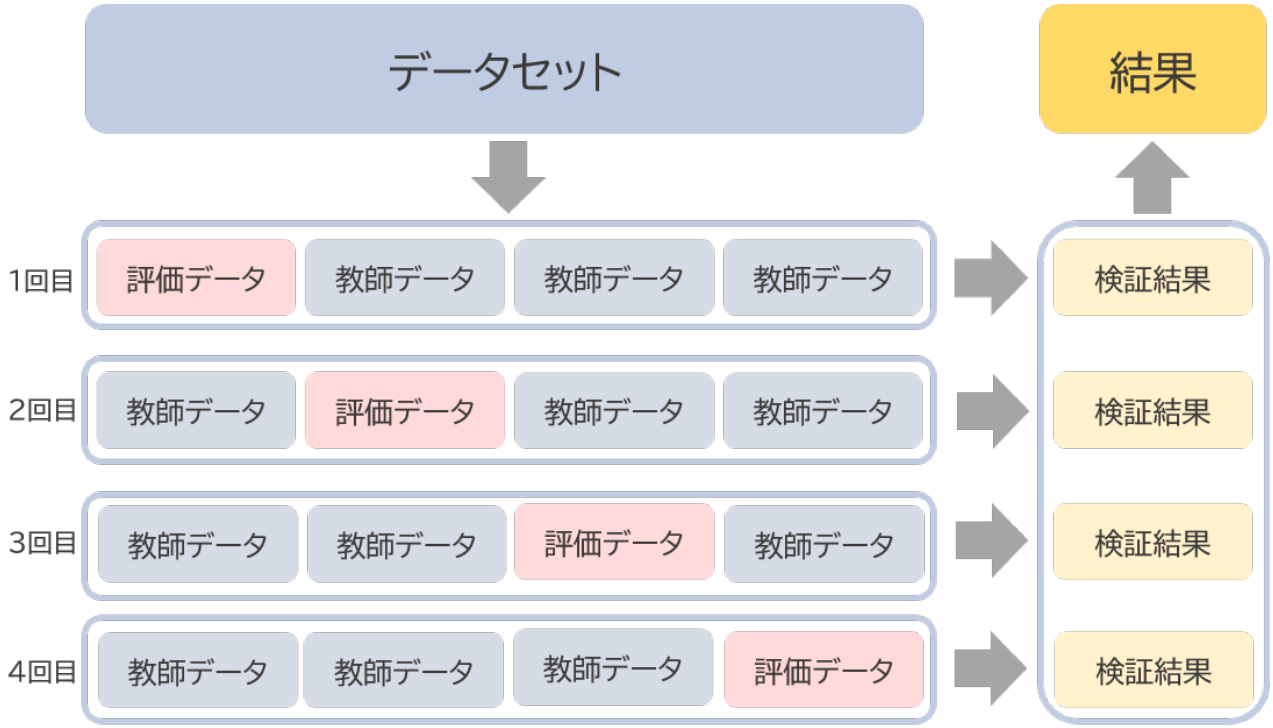
どの教師あり学習の手法にも評価方法があり、その値に基づいてモデルの評価が行われます。例えば分類を目的とした教師あり学習では、教師データで作成した予測モデルで評価データの分類を行ったときに何件中何件当たっていたのかが評価の対象になりますし、回帰を目的とした教師あり学習では、教師データで作成したモデルで評価データの回帰を行ったときに、教師データの予測値と評価データの実測値の差分がどのくらいなのかが評価の対象になります。出力された分類の正答率や回帰の値など予測結果の評価値を見て、そのモデルが実際の業務に使用できそうなモデルかどうかを検討し、使うことが難しそうであれば再度モデルを作り直す、といったトライアンドエラーの繰り返しによりモデルは作られていきます。

教師データと評価データにデータを分けて機械学習のモデルの評価を行う、そのために交差検証法は忘れずに行う必要があります。交差検証法の代表的な手法として「ホールドアウト法」と「k分割法」があります。

次項ではこれらの手法を詳しく見ていきます。

交差検証の方法：K分割法

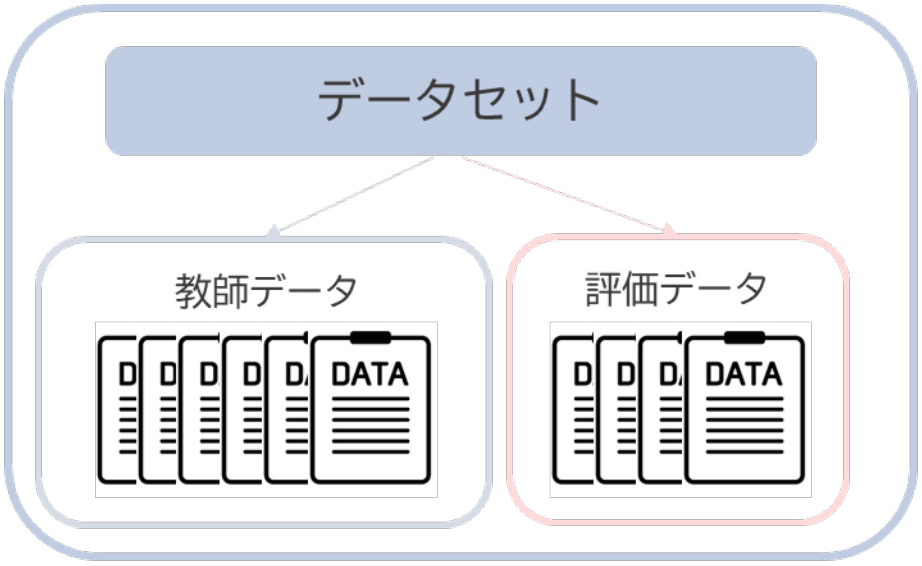
K 分割法は、学習データをいくつか（k個）に分割してモデル構築と検証を分割したデータの回数分（k回）繰り返す交差検証方法です。分割したデータ群のうちの 1 つを検証用とし、残りのデータでモデル構築を行います。1 つを検証用として、残りのデータで構築したモデルの当てはまり具合を見て 1 回目の結果を確認し、2 回目は別のデータを検証用として、残りのデータでモデル構築を行い・・・というように、分割したデータの個数分だけモデル構築と検証を行い、複数回（k回）繰り返し実施した検証の平均値を結果とします。これは異なるデータを検証用として複数回（k回）試行することにより、特定のデータのみに適合したモデルになってしまうことを防ぎます。



データを分割した回数分のモデル構築を行い検証結果の平均値を結果とする（k分割法）

データ分割方法：ホールドアウト法

ホールドアウト法とはモデルの精度を確認するための手法の一つで、1つのデータセットを教師データと評価データに分けて行います。モデルの精度に影響を及ぼしてしまうため教師データと評価データは必ず分けて実施する必要があります。ホールドアウト法では、教師データに使用したデータは評価データとして使うことはありません。逆に評価データとして使用したデータを教師データに用いることもありません。ホールドアウト法を用いる場合には、データが混同しないように注意することが必要です。



データを教師データと評価データに分割してモデル構築を行う
（ホールドアウト法）

教師あり学習の手法と活用例

「教師あり」の学習手法において、実際にどのような手法があるのでしょうか。
それぞれの手法について、具体的な活用例もあわせて紹介します

線形回帰分析

線形回帰とは、回帰分析の一種で、ある目的変数の値を、別の説明変数の値に基づいて予測する手法のことです。予測したいデータ・値を目的変数、予測するために使用するデータのことを説明変数と言います。

Case **線形回帰を使って、人気ケーキショップの需要予測をしてみよう**

美緒さんの両親は、市内に 10 店舗をもつケーキショップのオーナーです。美緒さんのおじいさん・おばあさんの代からはじめた小さなケーキショップでしたが、パティシエ修行をしていた美緒さんのお母さんがパリのコンクールで出会ったお父さんと結婚したことをきっかけにお店を継ぎ、この20年間で市内に10店舗あるお店にまで成長しました。

美緒さんもときどき本店でアルバイトとして、お客さんに商品について説明をしたり、レジで会計の手伝いをしています。
近所でとてもおいしいと評判のお店ですが、ここ半年ほどはケーキが売れ残ってしまうことが多くなっています。
とてもよく売れるので、「多めに作っておこう」とお母さんが言い始めたためです。

余りがたくさんでてしまっても、日持ちのしないケーキは翌日販売することができません。
「勿体ない」と思いながらも廃棄しなければならず、大好きなお店のケーキを捨てなくてはならないことに美緒さんはとても悲しく思ってました

美緒 「1 日どのくらいケーキが売れるのかわかればいいのに・・・」

「そういえば、お店の公式 Instagram に写真をアップしたり、季節の果物をテーマにした新作のケーキを販売する日は、お客さんが沢山来る気がする！」

「バレンタインやホワイトデー、クリスマスは毎年とっても忙しいし。

毎日ケーキがどれだけ売れるかが事前に分かれば、余らせてケーキを捨てなくても済むんじゃないかな！」

美緒さんはふと、学校で習った機械学習の授業のことを思い出しました。

「そういえば、事前に商品がどのくらい売れるのかを分析する「需要予測」について習ったな。私にもできるかな・・・？」

単回帰分析

1つの目的変数を1つの説明変数で予測するものを「単回帰分析」といいます。その予測を行う2つのデータの関係性は$y = ax + b$という一次方程式の形で表せます。これは「回帰」で使われる最も基本的なモデルです。

単回帰分析 ： $y = ax + b$

例：Instagram の投稿数からケーキの売上金額を予測する
目的変数：ケーキの売上金額　　説明変数：Instagram の投稿数

重回帰分析

2 つ以上(2次元以上)の説明変数を持つものを「重回帰分析」といいます。適切な変数を複数選択することで、計算しやすく誤差の少ない予測式を立てることができます。

重回帰分析 ： $y = a1x1 + a2x2 + a3x3 + \cdots\cdots + b0$

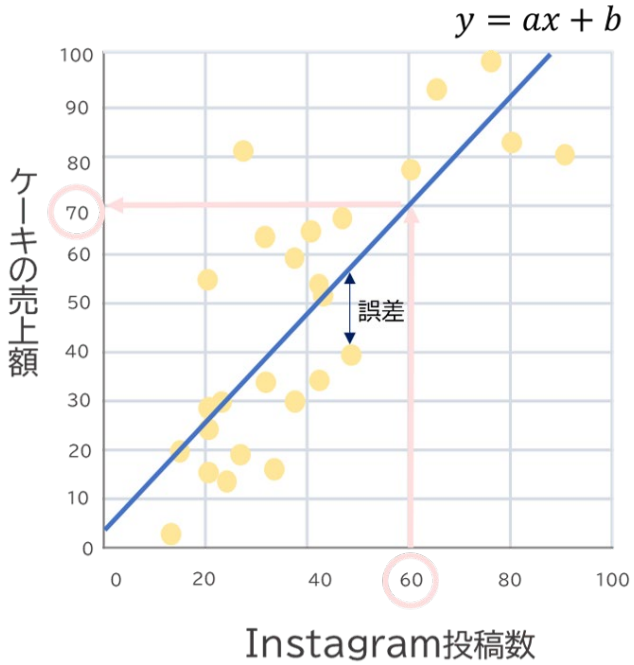
例:ケーキの売上金額を Instagram の投稿数・チラシの配布枚数、イベントの実施内容から予測する

目的変数:ケーキの売上金額

説明変数:Instagram の投稿数、チラシの配布枚数、季節のフルーツを用いたケーキの発売日(説明変数が複数)

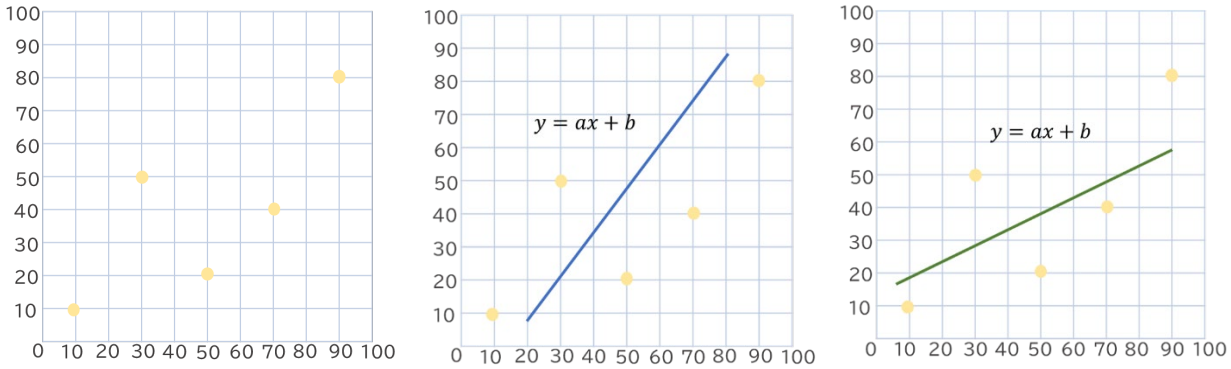
回帰分析の意味

『回帰』は教師あり機械学習の基礎の基礎です。回帰分析のうち、単回帰分析は 1 つの目的変数を 1 つの説明変数で予測するもので、2 つのデータ間の関係性を$y = ax + b$という一次方程式の形で表しますa(傾き)とb (Y 切片)が判明すれば x (Instagram の投稿数)から y(ケーキの売上金額)を予測することができます。このように 1 次方程式で表されることから「線形」と呼びます。また、予測の精度は相関係数(決定係数)で表します。



回帰直線をどのように引くか

2 つのデータを散布図に配置した際、目的変数(ケーキの売上金額)に対して予測できる線、回帰直線はどのように引くのが良いのでしょうか。いろいろなパターンが考えられますが「予測」がどのように行えればよいかを考えてみましょう。Instagram の投稿数からケーキの売上金額を予測する数式が作れたとき、予測した値が大きく外れていては意味がありません。しかし、ピタリと当たるような「直線」を引くこともできません。そこで、この「直線」を使って予測を行なった際に、実際の値(実測値)と最も「誤差」が少なくなるように線を引くことを考えます。**「予測した値と実測の誤差の二乗の和を最小にする」**計算方法を用いて、回帰直線の傾きと切片を求めていきます。この方法を**「最小二乗法」**と言います。この「誤差を最小にする」考え方は、説明変数が増えた重回帰分析でも全く同じです。



最小二乗法は、誤差の二乗の和を最小にするという意味で、直線から各値の長さを 1 辺とした正方形の面積の総和が最小になるような直線を探すことになります。

$$誤差の二乗の和 = \sum_{i=1}^{n} (Y_i - \alpha X_i - b)^2$$

回帰係数はどのように求めるか

回帰分析は予測をすることが目的のひとつです。相関関係を$y = ax + b$ の一次方程式で表せたとすると、定数の a (直線の傾き)と b (y軸の切片)がわかっていれば、x(Instagram の投稿数)からy (ケーキの売上金額)を予測することができます。

傾きは以下の式で表されます。

$$直線の傾き = \frac{[\mathrm{X}とY\text{の共分散}]}{[\mathrm{X}\text{の分散}]} = \frac{\frac{1}{n}\sum_{i=1}^{n}(Xi - \bar{X})(Yi - \bar{Y})}{\frac{1}{n}\sum_{i=1}^{n}(Xi - \bar{X})^2} = \frac{\sum_{i=1}^{n}(Xi - \bar{X})(Yi - \bar{Y})}{\sum_{i=1}^{n}(Xi - \bar{X})^2}$$

傾きを求めることができれば、y 切片 b を求めることができます。回帰直線は、(xの平均，y の平均)を通ることが分かっているので、以下の式からbが求まります。

$$b = \quad y\text{の平均} - \text{傾き} \times x\text{の平均}$$

例題で計算してみよう

	X	Y				
店舗	Instagram の投稿数	1 年間のケーキの売上金額(万円)	③X の平均との差	④Y の平均との差	⑤[X の平均との差③]と[Y の平均との差④] の積	⑥X:平均との差の二乗 (③の二乗)
西国分寺	5	240				
大泉学園	10	750				
経堂	3	110				
東雲	6	240				
江戸川橋	11	740				
上大岡	7	210				
代々木	15	580				
池袋	20	300				
吉祥寺	1	80				

① X の平均 ［　　］ ② Y の平均 ［　　］

共分散:⑤の平均 ･･･ ⑥

⑥X:平均との差の二乗の平均(X の分散)･･･⑦ ［　　］

⑧直線の傾き ⑥÷⑦ ･･･⑧

⑨切片 ･･･ ② - ①×⑧ ･･･ ⑨

回帰式 ： ⑧ x ＋ ⑨

例題で機械学習を実行しよう（Python を用いた機械学習の実行）

ここからの例題では、日別の売上金額と様々な説明変数のデータセットを用いて回帰分析を実行します。

回帰分析

```
[ ] #1 ライブラリのインポート
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    %matplotlib inline
```

データを読み込みます。

データ内容は以下の通り

ID：ID
sales：売上個数
insta_post：Instagram投稿数
insta_good：Instagramいいね！数
flyer：チラシ配布枚数
event：イベントあり＝1、イベントなし＝0
new_item：新作発売日＝1
holiday：土日祝＝1

```
[ ] #2  データを読み込みます。
    df = pd.read_csv("線形回帰.csv", index_col=0)
    df.head()
```

ID	sales	insta_post	insta_good	flyer	event	new_item	holiday
1	62	0	20	0	0	0	0
2	60	0	24	0	0	0	0
3	104	0	26	0	0	0	0
4	102	0	22	0	0	0	0
5	178	0	39	0	0	0	1

単回帰分析

説明変数をinsta_good、目的変数をsalesとして、単回帰分析を行います。

```
[ ] #3 説明変数insta_good
    X = df.iloc[:, 2].values
    X = X.reshape(-1,1)

    # 目的変数sales
    Y = df.iloc[:, 0].values
```

```
[ ] #4 sklearn.linear_model.LinearRegression クラスを読み込み
    from sklearn import linear_model
    clf = linear_model.LinearRegression()

    # 予測モデルを作成
    clf.fit(X, Y)

    # 回帰係数
    print(clf.coef_)

    # 切片
    print(clf.intercept_)

    # 決定係数
    print(clf.score(X, Y))
```

```
[2.05852128]
101.71515440413765
0.5218980221448157
```

insta_goodの回帰係数と、単回帰の式の切片が分かりました。
決定係数は、0～1の範囲の値をとり、値が大きいほどモデルが適切にデータを表現できているといえます。

結果の解釈

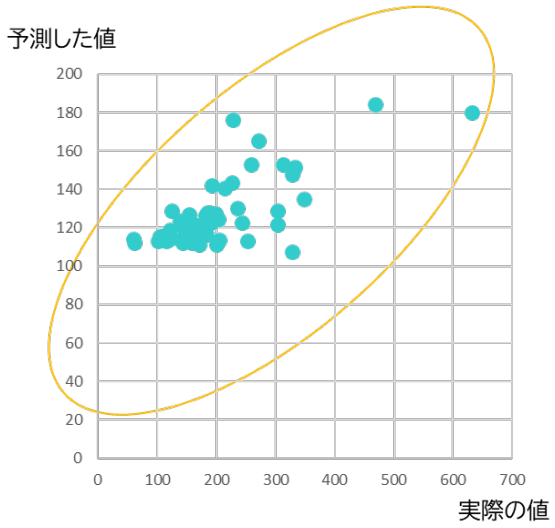
回帰式の係数と切片を算出することが出来ました。上記の例では、$y = 2.059x + 101.7$ という数式を用い、insta_good(Instagram のいいね！数)がわかれば、2.059 を掛けて 101.7 を加えると売上金額が予測できるということがわかりました。では、その予測はどのくらい実際の値と違っているのでしょうか？ それを把握

するために、この例では「決定係数」を算出しています。

相関係数・決定係数とは

相関係数は、あるデータともう 1 つのデータ、2 つのデータの関係性を把握するための指標で、偏差の積和の平均(共分散)を X と Y の標準偏差で割ったものが相関係数です。相関係数は-1 から 1 までの値をとり、絶対値が 1 に近ければ近いほど 2 つのデータの関係性が高いとみなします。回帰分析の実行結果を評価する際には、まず「回帰式により求まった値」と「実際の値」の相関関係を確認します。また、決定係数は相関係数の二乗の値です。二乗することによりマイナスの符号もプラスの符号も関係なく評価ができるようになります。「回帰式により求まった値」と「実際の値」の相関係数・決定係数が 1 に近ければ近いほど、回帰式による予測が出来ていると考えることができます。

$$\text{相関係数} = \frac{[\text{XとYの共分散}]}{[\text{Xの標準偏差}]\,[\text{Yの標準偏差}]}$$



ロジスティック回帰とは

ロジスティック関数を用いた回帰モデルのことです。線形回帰モデルとの大きな違いは目的変数の種類にあります。線形回帰モデルは目的変数に実数値をとりますが、ロジスティックス回帰はカテゴリや比率をとります。つまり、あるサンプルにラベルを与えるような分類問題で用いられるのです。

分類の際はデータのプロット上に境界線を引くことを考えます。回帰によって各点にフィットする線ではなく、各点を分割する線を求めるということです。この手法で目的変数と説明変数をつなぐ関数として、ロジスティック関数を用います。この関数に分類対象のデータを入力すると、0 から 1 の値をとる確率を返します。

問題ごとに適切な閾値を定めて、その値と出力された確率の大小によってクラスの分類を行うのです。活用例としては、たばこの本数と肺疾患の関係、クレジットカードの保有者情報と債務不履行の関係、という問題があげられます。このように二つの選択肢の答えを考える場面が実際の社会では多く、有用な手法です。

判別分析

判別分析は二つの母集団を設定し、あるサンプルがどちらの母集団に属するのかを推測するための手法です。母集団への所属があらかじめわかっているサンプルとその変数について判別方式を構成し、その方式を用いて所属不明なサンプルの属するクラス母集団を求めることができます。

判別方式としては、変数の値から各母集団への距離を定義し、対象のサンプルがその距離の最も小さい母集団へ属するという考えをとります。

そこで用いる距離は**マハラノビスの距離**と呼ばれ、各母集団における変数の確率分布が正規分布にしたがうとする前提と対応しています。

活用例としては、世論調査から選挙の当選者の予測、模擬試験結果からの合格予測、身元不明の骨から年齢や性別を予測、顧客の傾向から将来の発注予測などがあげられ、適用範囲の幅広さで知られています。

数量化理論(I 類、II 類)

数量化理論の「数量化」とは、**数量でないデータ(質的データ)を数量として分析する**という意味で命名されたもので、林知己夫によって開発された日本発の多変量解析手法です。

アンケートデータのクロス集計表を可視化するのに用いられるコレスポンデンス分析等の手法がそれにあたり、コンピュータが現在ほど進化していない時代には画期的手法でした。この手法は現在でも広く活用されています。

数量化理論には I 類から VI 類までありますが、よく用いられるのは I 類から IV 類までで、V 類や VI 類はほとんど使われていません。数量化 I 類と II 類が、機械学習において「教師あり学習」に分類されるのは、目的変数があるデータを分析するからであり、多変量解析においては予測の手法と要約の手法のうち、予測の手法にあたります。

数量化 I 類とは

数量化 I 類は、データが性別や学年のようにグループに分かれている場合に、その性別や学年による性質が、どの程度「量」としてに違うかということを分析する手法です。数量化は、質的データを数量化して分析するため、性別や学年のように数値でないデータ(カテゴリデータと言われます)を説明変数として、量的変数である身長や体重がグループ毎にどう異なるかという事を知りたい時に使います。

より具体的に説明すると、ある人が高校 1 年生の女性だった場合に、その人の身長と体重を予測するモデルを作ることができます。重回帰分析が、身長(量的データ)から体重(量的データ)を予測するモデルを作るのに対して、数量化 I 類では、質的データから量的データを予測することができます。

数量化Ⅱ類とは

数量化 I 類は、説明変数が質的、目的変数が量的であったのに対し、数量化 II 類は説明変数も目的変数も質的である場合に使います。

たとえば、商品を販売するための広告データなどを分析する際に、[商品の色][形][ブランド]などの特徴を表わす説明変数と、[ある商品を購入した／しない]、[広告をクリックした／しない]という目的変数のデータがあった時に、説明変数から目的変数を予測したり、説明変数と目的変数の関係を調べたり、分析結果を可視化したりすることができます。

数量化Ｉ類の分析アルゴリズムは、目的変数が量的変数である**重回帰分析**に対応し、数量化 II 類は**判別分析**や**ロジスティック回帰分析**に対応しています。
数量化理論においては、性別や年代などの質的変数をダミー変数という二値変数に変換して分析をしているので、目的変数をダミー変数に変換した重回帰分析等の多変量解析と同じ手法とみなすこともできます。

		（結果）目的変数	
		量的変数	質的変数
原因（説明変数）	質的変数	**数量化Ｉ類** 性別と果物の好き嫌いからケーキの消費量を予測	**数量化Ⅱ類** 性別と果物の好き嫌いから、ケーキが好きかどうかを予測
	量的変数	**重回帰分析** Instagram 投稿の「いいね！」数から、ケーキの販売額を予測	**判別分析、ロジスティック回帰分析** ケーキの消費量から性別を予測

表 1：変換前（数量化前）のデータ

性別	学年	身長（cm）	体重（kg）
男性	高校１年生	177	61
女性	高校２年生	162	68
女性	高校３年生	165	70
男性	高校１年生	169	61
男性	高校３年生	172	68
女性	高校２年生	158	66
女性	高校１年生	162	53
男性	高校３年生	171	59
男性	高校１年生	169	63
男性	高校３年生	169	65
女性	高校２年生	172	58
女性	高校１年生	158	54

表2：変換後のデータ

女性	高校１年生	高校２年生	身長(cm)	体重(kg)
0	1	0	177	61
1	0	1	162	68
1	0	0	165	70
0	1	0	169	61
0	0	0	172	68
1	0	1	158	66
1	1	0	162	53
0	0	0	171	59
0	1	0	169	63
0	0	0	169	65
1	0	1	172	58
1	1	0	158	54

ここで、表2の表頭に男、高校3年生が抜けているのは、女性が 0 の場合は男性、高校1年生でも高校2年生でもない場合は高校3年生ということが明らかなので、同じ意味のデータが含まれていると多重共線性といわれる相関の高いデータがあった時に計算上の不都合が起こる問題があるからです。

サポートベクターマシン（SVM）

2つのクラスのパターン識別を構成する手法です。訓練データから、「各データとの距離が最大となるマージン最大化超平面を求める」という基準で線形入力素子のパラメータを学習させます。

Case　スマートフォンの故障は予測できる？

美緒さんは高校のお友達との間で、部活の合宿期間後から、スマートフォンが故障して困っているという話が増えています。あまりにいろんな人のスマートフォンが壊れるため、そのうち「スマホが壊れる人は呪われている」という噂までたち始めました。

美緒さんのお友達は怖がりで、とても気にしています。

そこで美緒さんは、「呪いなんかのはずがない。壊れるには原因があるはずだ」と考え、それをどうにか解明できないかを考えました。

「そういえば、ポケットにスマホを入れたまま合宿に参加して、落としている子がよくいたな。」

もしかしたら、機種によっても違いがあるかもしれない。」

「つい最近壊れたと言っていたのは、中学からスマホを持っている子だったな。使用年数が長いと壊れやすくなるのかも。」

壊れる時期や壊れやすいスマートフォンを、あらかじめ知ることはできるのかな？

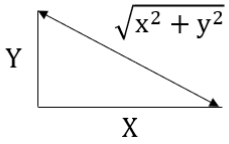
美緒さんはふと、学校で習った機械学習の授業のことを思い出しました。

「そういえば、対象を2つに分割する「サポートベクターマシン」について習ったな。

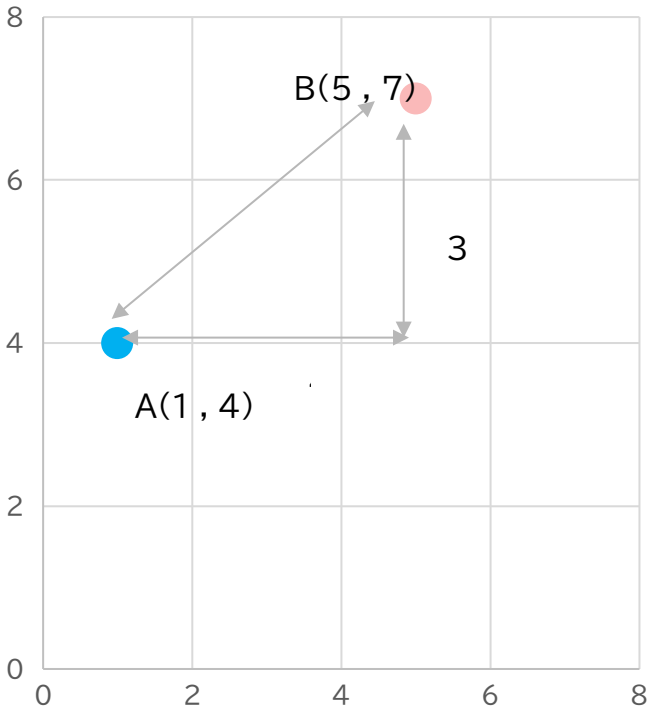
私にもできるかな・・・？」

サポートベクターマシン（SVM）が行うのは、「境界を識別する関数を求める」ということです。この「識別面」から最も近い距離サンプルまでの距離（マージン）をとり、マージンを最大化する識別面を決定します。
そして、識別面に最も近いサンプルのことを「サポートベクトル」といい、それを用いて境界を決定します。SVMは、次元（説明変数）が多くても対応でき、パラメータが少ないため使いやすいと言われています。このとき、距離の計算方法には「ユークリッド距離（二点間の距離）」が使われるのが最も一般的です。ユークリッド距離は、二点間の距離のことで、ピタゴラスの定理で表すことができます。

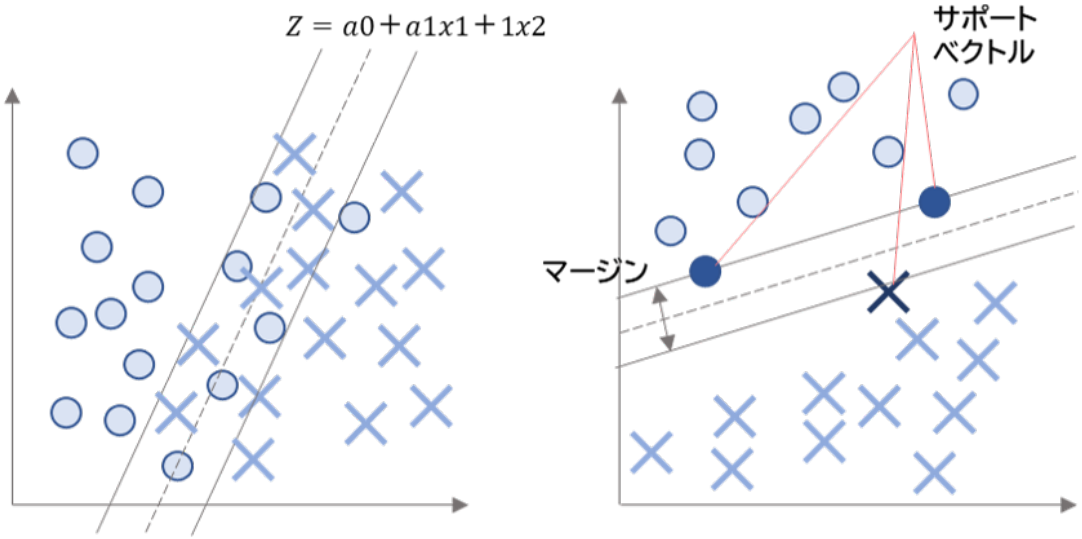
$$d(x,y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$$



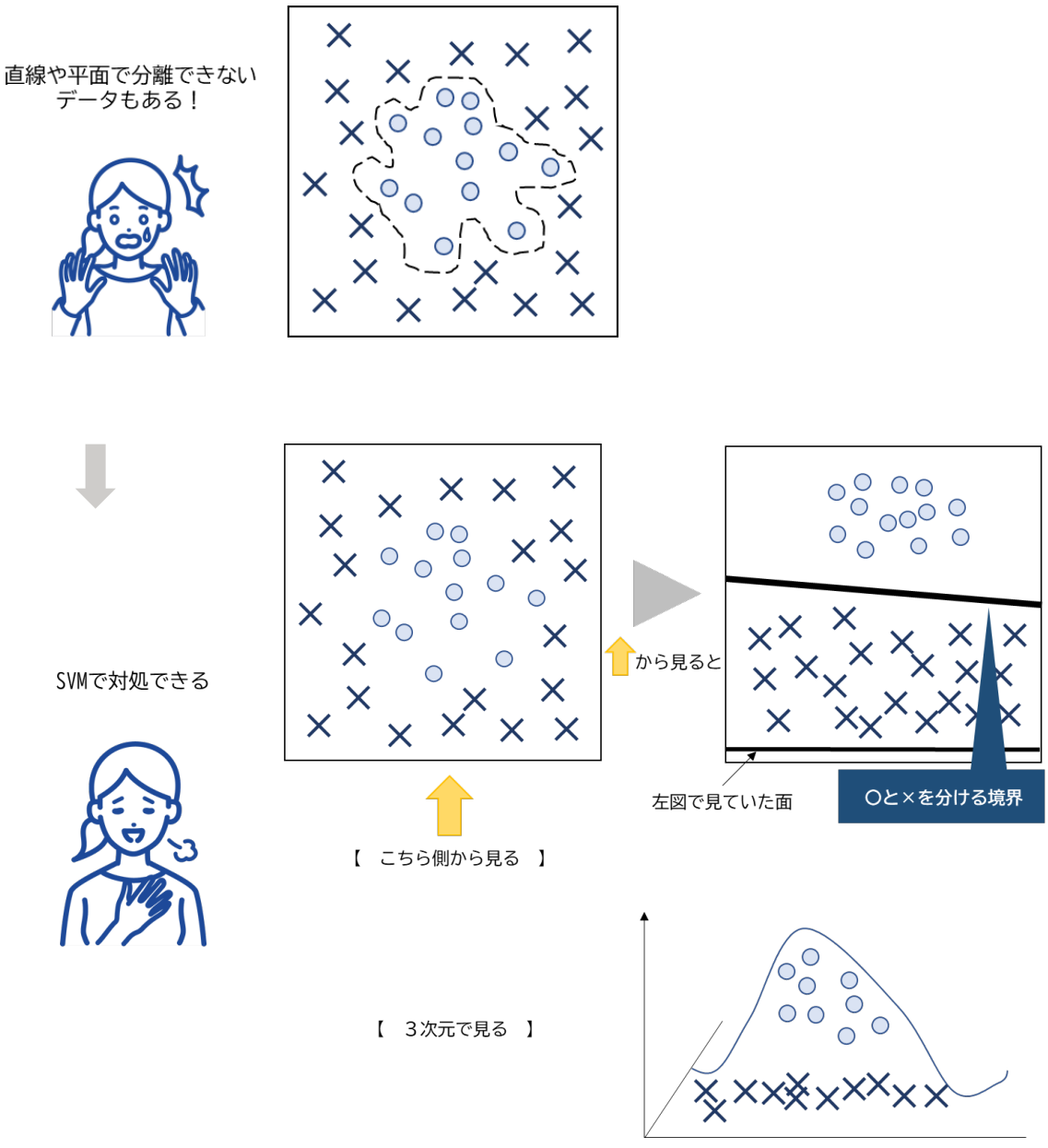
A さん(1,4)と B さん(5, 7)の距離を計算してみましょう。



A さんと B さんの距離は、三平方の定理を用いて 5 と求めることができます。サポートベクターマシンでは、このようにして「距離」を図り、マージンを最大化できるところに境界線を引いています。



直線や平面でシンプルに分離できない・識別できないデータ（説明変数が多数で多次元のときなど）であっても、SVM はそのデータをその高次元空間上で線形に変換して捉える処理を行う「カーネルトリック」によって　、高い精度で識別する　ことが可能です。



例題で計算してみよう

使用するデータセット

ID	使用月数	落とした回数	性別	年代	故障歴の有無	ヒビの有無	機種の幅	機種の高さ	機種の厚さ	カバーの有無	スマホリングの有無
1	38	0	1	10	0	0	60	140	7	1	0
2	23	0	2	10	0	0	60	140	7	1	1
3	3	2	1	10	1	1	64	131	7	0	0
4	30	0	1	10	0	0	78	160	7	0	0

例題で機械学習を実行しよう（Python を用いた機械学習の実行）

▾ SVM

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

データを読み込みます。

データ内容は以下の通り

ID：ID
months：使用月数
falls：直近1か月で落下させた回数
sex：利用者の性別（1=男性／2=女性）
age：利用者の年代
past_falls：過去の故障歴の有無
crack：ヒビの有無
width：機種サイズ（幅）
height：機種サイズ（高さ）
thick：機種サイズ（厚み）
cover：カバーがついているか
ring：リングがついているか

```
[2] #2
df = pd.read_csv("SVM.csv",index_col=0)
df
```

7	5	1	2	10	0	1	71	146	7	0	0
8	5	0	2	10	0	1	78	160	7	1	0
9	36	1	1	10	0	0	67	138	7	1	0
10	35	1	2	10	0	0	77	158	8	1	0
11	32	0	2	10	0	0	71	144	8	1	0
12	38	0	2	10	0	0	71	144	8	1	1
13	12	2	1	10	1	0	73	150	7	1	1

スマホが壊れたかどうかを目的変数とし、故障に関係のありそうな変数を選んでモデルを作成してみます。

```
[3] X = df.drop(['past_falls','sex','age','ring'], axis=1).values

    # 目的変数
    Y = df['past_falls'].values
```

```
[4] from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler

    #学習データとテストデータに分割
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0)

    # データの標準化処理
    sc = StandardScaler()
    sc.fit(X_train)
    X_train_std = sc.transform(X_train)
    X_test_std = sc.transform(X_test)
```

PCAを使用して、次元を削除します。
多次元データを2次元データに減らしています。

```
[5] #3
    from sklearn.decomposition import PCA
    pca = PCA(n_components = 2)
    X_train_std2 = pca.fit_transform(X_train_std)
    X_test_std2 = pca.fit_transform(X_test_std)
```

```
[6] #4
    X_train_std.shape
```

```
    (27, 7)
```

```
[7] #5
    X_train_std2.shape
```

```
    (27, 2)
```

モデルの構築

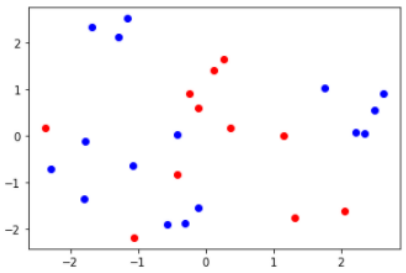
```
[8] #6
    from sklearn.svm import SVC

    #モデルの構築（線形SVM，RBFカーネル）
    model1 = SVC(kernel='linear')
    model2 = SVC(kernel='rbf')

    model1.fit(X_train_std2, Y_train)
    model2.fit(X_train_std2, Y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

分類結果の図示

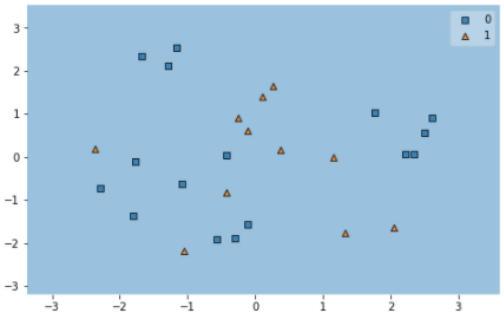
実際に、学習データのモデルを用いて分類結果を図示します

```
[9] #7
    from matplotlib import pyplot

    pyplot.plot(X_train_std2[Y_train == 1,0], X_train_std2[Y_train == 1,1], 'ro')
    pyplot.plot(X_train_std2[Y_train == 0,0], X_train_std2[Y_train == 0,1], 'bo')
    pyplot.show()
```




```
[10] #8  分類結果を図示する
    import matplotlib.pyplot as plt
    from mlxtend.plotting import plot_decision_regions

    fig = plt.figure(figsize=(8,5))
    plot_decision_regions(X_train_std2, Y_train, clf=model1)
    plt.show()
```

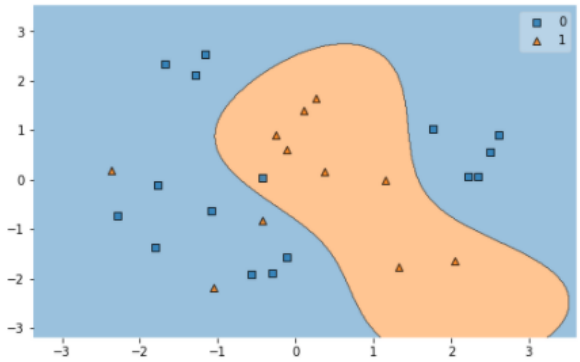
```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:242: UserWarning: No contour levels were found within the data range.
  antialiased=True)
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244: MatplotlibDeprecationWarning: Passing unsupported keyword arguments
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```



線形では上手く分離できていないことが分かりました。

```
[11] #9  分類結果を図示する
     import matplotlib.pyplot as plt
     from mlxtend.plotting import plot_decision_regions

     fig = plt.figure(figsize=(8,5))
     plot_decision_regions(X_train_std2, Y_train, clf=model2)
     plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244: MatplotlibDeprecationWarning: Passing unsupporte
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```



RBFカーネルでは、上手く分けられているように見えます。

モデルの評価

SVM による予測と、実際の結果(正解・教師)はどうだったでしょうか。結果が合っていたか、合っていなかったかは、下記の表(混同行列といいます)で表すことができます。

		実際	
		故障	故障しない
SVM による予測	故障	【A】10 正解	【B】0 不正解
	故障しない	【C】1 不正解	【D】80 正解

データサイエンティストが実際に予測を行った場合は、下記のそれぞれの指標で確認します。

Accuracy(正答率): $\frac{A+D}{A+B+C+D}$ Recall(再現率): $\frac{A}{A+C}$ Precision(精度): $\frac{A}{A+B}$

F-Measure(F値): $\frac{2Recall \cdot Precision}{Recall+Precision}$

決定木

決定木は機械学習の手法のひとつで、デシジョンツリー(decision tree)とも呼ばれます。
樹形図、ツリー構造型の図を作り出し、分類を行なうためのルールを作成する分析手法です。最終結果として分岐条件が提示されるため、分析結果を実務に適用し易い手法です。1963　年に Morgan & Sonquest が提唱し、予測と分類に多く用いられています。

Case　常連顧客になる・ならないは何が違う？

決定木は、「予測」「分類」を多段階で実行する教師あり機械学習の手法です。目的変数がある探索的分析手法で量的変数、質的変数どちらも扱えます。また、分岐過程を階層化し樹形図で表現するので視覚的にたいへんわかりやすい手法でもあり、非線形データにも対応可能です。

最近、美緒さんはアルバイトをしながら、ケーキショップの売上をもっと伸ばしていくためにはどうしたらいいか？ について考えるようになりました。

「新しいお客さんにお店のことを知ってもらうのも大切だけど、常連さんがもっと増えたらいいな」
お店には多くの顔なじみのお客さまがいます。昔から贔屓にしてくれる人もいて、イベントがあるときは必ず美緒さんのお店のケーキを購入してくれます。

「常連になってくれるお客さまとそうでないお客さまは、何が違うんだろう？」
ふと美緒さんは思いました。
しかし、アルバイトである美緒さんにはレジで接客したときに話すお客さまの情報しかありません。
もっと今の常連客のことを知る方法があれば･･･。

ふと、美緒さんはあるものを思い出しました。お店では会員に発行しているポイントカードです。
ポイントカードには、お客さんごとにこれまでの購入履歴の情報がシステムに記録されています。
しかし、これまではポイントカードの情報をお店で活用したことがありませんでした。
ポイントカードには、ほかにどんなデータがあるんだろう？
お店の売上をあげるためのヒントを見つけられないかな･･･？

そこで美緒さんは、ポイントカードのデータと POS データを活用し、
常連客を増やすためのヒントを得られないかと考えました。

「そういえば、お会計の時にお客さまと話していると、結構遠くから買いに来てくれている方が多い気がする。」
「季節の果物をテーマにした新作ケーキは、一見さんに多いかも！」
「通年販売のケーキはアフタヌーンティーを楽しむマダムたちに人気な気がする」

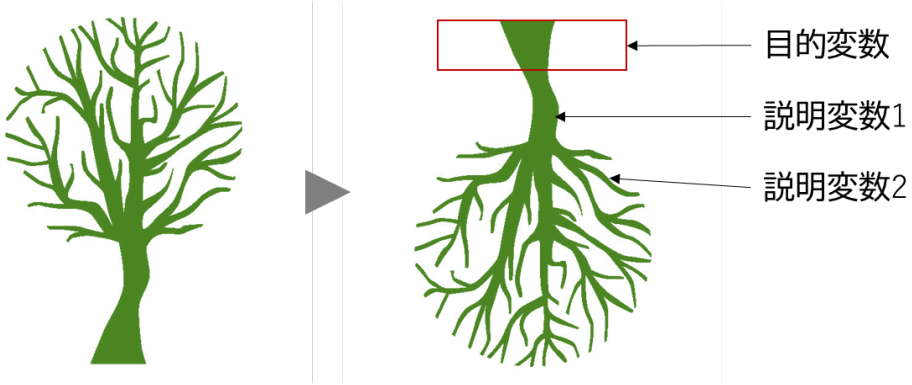
考えれば考えるほど、常連客に必要な条件は複雑に思えてきます。これを知る方法はあるのでしょうか？

そこで美緒さんはふと、学校で習った機械学習の授業のことを思い出しました。
「そういえば、条件ごとに対象者を分類できる「決定木」について習ったな。
あの方法だったら、常連客の条件が見つけられるかも。私にもできるかな･･･？」

樹形図(ツリー)によって分岐する条件が視覚化されることから「説明が求められる」場合の機械学習の手法としてビジネスのいろいろな場面で活用されています。決定木には分類木と回帰木の 2 種類があり、分類木は対象の分類を行いたい場合、回帰木は対象の連続する数値を予測したい場合に用いられます。

目的変数と説明変数を設定し、目的変数の特徴的なグループを観測データに見出していきます。複数の説明変数による条件でデータを分割すると目的変数の特徴がはっきりしてきます。決定木とは、このように説明変数の条件で構成されたデータの分岐ルールを生成する手法です。

また、決定木ではその手法の特性から過学習しやすい傾向があります。

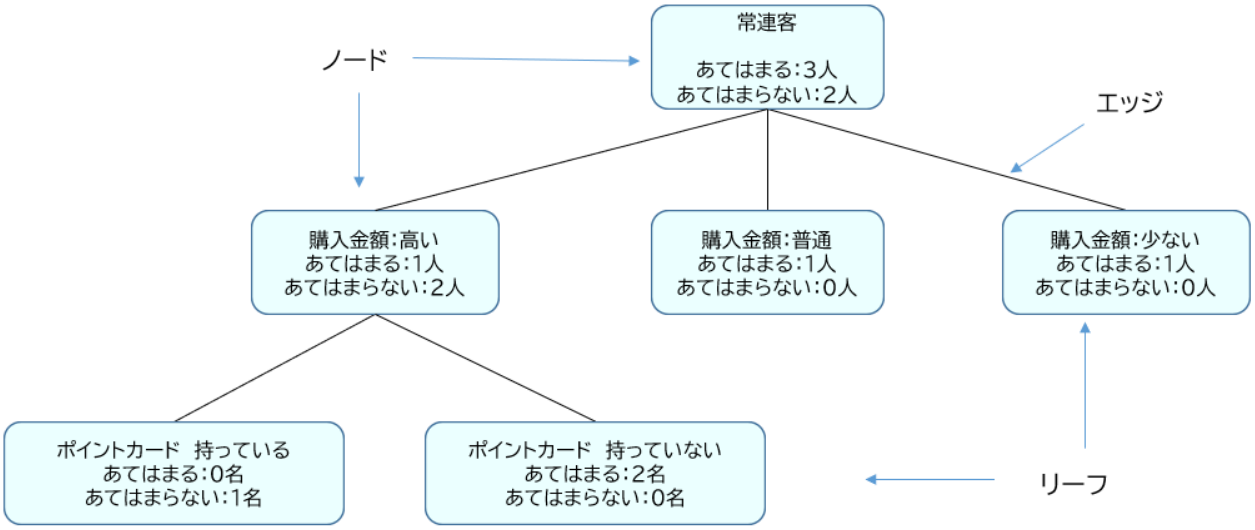


分類木　例:常連顧客になる/常連顧客にならない
回帰木　例:ケーキの販売額を決める要因はなにか？

データが下記のようなケースを考えてみましょう。

顧客番号	常連客である	購入金額	ポイントカードの所持
A	あてはまる	多い	持っている
B	あてはまる	普通	持っていない
C	あてはまらない	普通	持っている
D	あてはまる	多い	持っていない
E	あてはまらない	少ない	持っていない

このようなケースでは、決定木は下記のように分岐します。



決定木では、目的変数と説明変数の候補となるデータをそれぞれ集計して分類し、最も分類される方法を繰り返し計算して見つけ、分岐する過程を目で見て捉えることができます。データのグループをノード、分岐が止まった一番末端にあるノードがリーフ、ノードとリーフを結ぶ線をエッジと呼びます。この分類を繰り返し計算するアルゴリズムには様々なものがあります。CHAID、C&R Tree などがよく知られています。CHAID はx^2値（カイ二乗値）と呼ばれる統計量が最も大きい順にデータを分割していきます。x^2値は、観測度数と期待度数のずれを表す数値で、x^2 分布という統計の分布を利用して「めったに起こらないこと」が起きるかどうかを計算する値です。CHAID はx^2値が大きなものから分岐させていくアルゴリズムです。x^2値を例題で計算してみましょう。

常連客にあてはまるか、あてはまらないかと、購入金額が高い・普通・少ないでクロス集計を行います。

		常連客 あてはまる	常連客 あてはまらない	合計
購入金額	**少ない**	1.0	0.0	1.0
	普通	1.0	0.0	1.0
	高い	1.0	2.0	3.0
	合計	3.0	2.0	5.0

常連客に「あてはまる」と、常連客に「あてはまらない」の比は3:2です。もし、購入金額が常連客で同じ比率であったとしたら、すべて3:2で下記の表のようになるはずです。これを期待度数といいます。

		常連客 あてはまる	常連客 あてはまらない	合計
購入金額	**少ない**	0.6	0.4	1.0
	普通	0.6	0.4	1.0
	高い	1.8	1.2	3.0
	合計	3.0	2.0	5.0

次に、期待度数と実際の値がどのくらい違うのかを算出します。期待されていた値から、実際の値を引き算します。

		常連客 あてはまる	常連客 あてはまらない	合計
購入金額	**少ない**	-0.4	0.4	0.0
	普通	-0.4	0.4	0.0
	高い	0.8	-0.8	0.0
	合計	0.0	0.0	0.0

符号がマイナスになっているものもあるので、比較しやすくするためにそれぞれの値を2乗します。

		常連客 あてはまる	常連客 あてはまらない	合計
購入金額	**少ない**	0.16	0.16	0.32
	普通	0.16	0.16	0.32
	高い	0.64	0.64	1.28
	合計	0.96	0.96	1.92

さらにその値を期待度数で割ります。カイ二乗分布に従うかどうかを計算するためです。

		常連客 あてはまる	常連客 あてはまらない	合計
購入金額	**少ない**	0.27	0.40	0.67
	普通	0.27	0.40	0.67
	高い	0.36	0.53	0.89
	合計	0.89	1.33	**2.22** **これがx^2値**

一方、ポイントカードを持っている／持っていない のx^2値は計算すると 1.88 です。購入金額の分岐のほうがx^2値の値が大きいため、先に分岐する条件になることがわかります。

例題で機械学習を実行しよう（Python を用いた機械学習の実行）

▼ 決定木分析

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

学習データの生成

```
[ ] # create sample dataset with 100 rows and 4 columns; regular_customer,  number_of_visits, pointcard, total_purchase_amount
    # regular_customer takes 1 or 0. If given cutomer is a regular customer, it takes 1. Otherwise 0.
    # number_of_visit takes natural number, representing how many times given customer has visited the shop.
    # pointcard takes 1 or 0. If given cutomer have a pointcard, it takes 1, otherwise 0.
    # total purchase is a sum of amount of all the purchase given customer has made in yen.

    # import dataset that will be base of new dataset
    from sklearn.datasets import load_breast_cancer

    cancer = load_breast_cancer()
    cancer_df = pd.DataFrame(cancer.data, columns = cancer.feature_names)

    # seed random number generator
    np.random.seed(1)

    # creating column vectors from cancer dataset
    regular_customer = np.array(cancer.target)
    number_of_visits = np.array(cancer_df['mean radius'])
    total_purchase_amount = np.array(cancer_df['mean texture']*100)

    point_card = []

    for i in range(len(cancer_df)):
      if cancer_df['mean concave points'][i] >= 0.05:
        point_card.append(0)
      else:
        point_card.append(1)



    # combining array
    my_array = np.transpose(np.round(np.array([regular_customer, number_of_visits, point_card, total_purchase_amount])).astype('int'))

    # convert array to pandas dataframe
    df = pd.DataFrame(my_array, columns=['regular_customer','number_of_visits','point_card', 'total_purchase_amount'])
```

```
cancer_df
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error	perimeter error	are erro
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	1.0950	0.9053	8.589	153.4
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	0.5435	0.7339	3.398	74.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	0.7456	0.7869	4.585	94.0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	0.4956	1.1560	3.445	27.2
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	0.7572	0.7813	5.438	94.4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	1.1760	1.2560	7.673	158.7
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	0.7655	2.4630	5.203	99.0
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	0.4564	1.0750	3.425	48.5
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	0.7260	1.5950	5.772	86.2
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	0.3857	1.4280	2.548	19.1

569 rows × 30 columns

学習用データを見てみます。

```
[ ] df = pd.read_csv("decision2.csv",index_col=0)
```

学習用データを見てみます。

[ ] df = pd.read_csv("decision2.csv",index_col=0)

```
[ ] df
```

regular_customer	number_of_visits_m	number_of_visits_s	number_of_visits_l	point_card	total_purchase_amount
0	1	0	0	1	0
0	0	1	0	0	1
1	0	0	1	1	1
1	1	0	0	0	1
1	0	0	1	0	1
...	...	...	...	...	...
0	1	0	0	1	0
0	0	1	0	0	1
1	0	0	1	1	1
1	1	0	0	0	1
1	0	0	1	0	1

75 rows × 5 columns

```
[ ] (point_card == regular_customer).all()
```

```
False
```

実際に決定木分析を行っていきます。

```
[7] from sklearn.model_selection import train_test_split

    #学習用データとテスト用データに分割
    X_train, X_test, Y_train, Y_test = train_test_split(
        df.drop('regular_customer', axis=1), df.regular_customer,  test_size=0.3)
```

```
[8] from sklearn.tree import DecisionTreeClassifier

    #決定木分析
    tree = DecisionTreeClassifier(max_depth=3, random_state=1)
    tree.fit(X_train, Y_train)
```

```
    DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                           max_depth=3, max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, presort='deprecated',
                           random_state=1, splitter='best')
```

学習用データとテスト用データの精度を確認します。

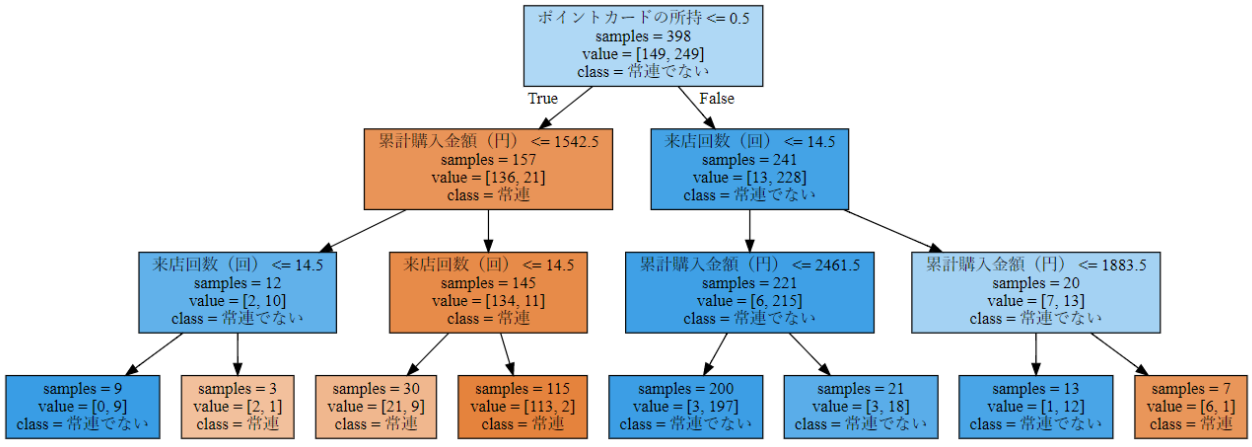
```
print("学習用データの精度：{:.3f}".format(tree.score(X_train, Y_train)))
print("テスト用データの精度：{:.3f}".format(tree.score(X_test, Y_test)))
```

```
学習用データの精度：0.950
テスト用データの精度：0.889
```

▾ 決定木の可視化

先程の決定木分析の結果を可視化します。

```
[ ] from sklearn.tree import export_graphviz
    import graphviz

    feature_names = ['来店回数（回）','ポイントカードの所持', '累計購入金額（円）']

    tree_data=export_graphviz(tree, out_file=None, class_names=["常連", "常連でない"],
                    feature_names=feature_names, impurity=False, filled=True)

    graphviz.Source(tree_data)
```



分析結果の解釈

予測の精度はどうなっているでしょうか。学習データでの精度が 0.95、テストデータでの精度が0. 889です。テストデータのほうが少し低いですが、いずれも高い精度となっています。これを「よく予測できている」と判断するか「過学習になっている」と判断するかはデータの取得背景やアルゴリズムへの理解も必要です。詳しくは「過学習」についての説明を確認してください。

生成された決定木の分岐図を確認してみましょう。今回の分析結果では、常連であるか／どうかの分岐条件としては、まずポイントカードの所持があるか否かで分岐していることが分かります。所持しているか・していないかの後は

機械学習を組み合わせる：アンサンブル学習

アンサンブル学習とは機械学習において、ロジスティック回帰や SVM、決定木など複数の学習モデルを組み合わせてより強力なモデルを構築する方法です。

アンサンブル学習には主に「バギング」と「ブースティング」 の2つの手法に分けられます。

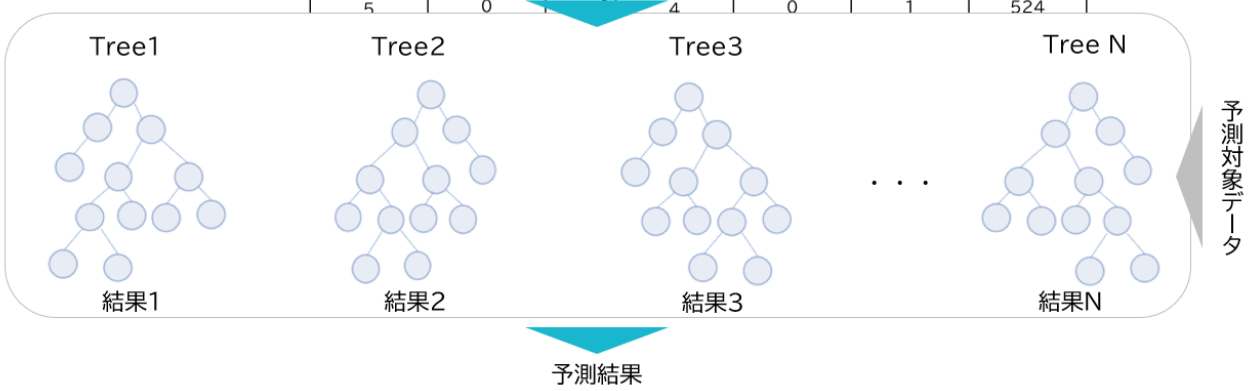
バギングとは、観測データからデータの一部をランダムサンプリング（ブートストラップ）し、複数の学習モデルを並列的に学習させることをいいます。ブースティングとは、複数の学習モデルを直列的に学習させることをいいます。

ランダムフォレスト

ランダムフォレストは、アンサンブル学習のバギングを基にして多数の決定木を用いて分類、回帰、クラスタリングを行う機械学習のアルゴリズムです。決定木は過学習しやすいという欠点を持っていますが、複数の決定木を用いることで克服することができます。シンプルでわかりやすく、精度も高いと言われる手法です。

学習用データ

顧客番号	常連かどうか	ポイントカードの所持	累計購入金額
0	1	1	1100
1	0	1	2100
2	0	0	22000
3	1	0	20040
4	0	1	524
5	0	0	



ニューラルネット

ニューラルネットの「ニューラル(neural)」は、ニューロン(神経細胞)のという意味で、ニューラルネットは生物の脳神経のメカニズムを人工的にコンピュータのプログラムで再現した、機械学習の手法の一つです。よく耳にする深層学習(ディープラーニング)は、このニューラルネットをより複雑にしたものです。

Case　修学旅行の写真を自動で分類しよう

美緒さんの学年は、つい先日いろいろな国をテーマにしたテーマパークに修学旅行に行きました。

カメラマンさんが撮影してくれた写真データを、美緒さんをはじめとする 5 人の実行委員で

写真データを整理するために国ごとに分けることにしました。

しかし、あまりにも枚数が多く、手作業だと大変です。

そこで美緒さんは「何とか自動で分けられないかな」と考えるようになりました。

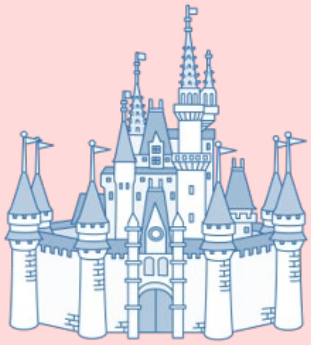
「何をヒントにしたら見分けられるだろう。イタリアぽいエリアとか、USA ぽいエリアとか、建物にも特徴があったし。むずかしそうだなあ。でも代表的な建物や風景なので、これらの国ごとの特徴を使って自動分類できるかも！」

美緒さんはふと、学校で習った機械学習の授業のことを思い出しました。

「そういえば、文章とか写真を入力するとどうにか分類してくれる

「ニューラルネットワーク」について 習ったな。

私にもできるかな･･･？」



ニューラルネットワークはなぜ画像を判定できる？

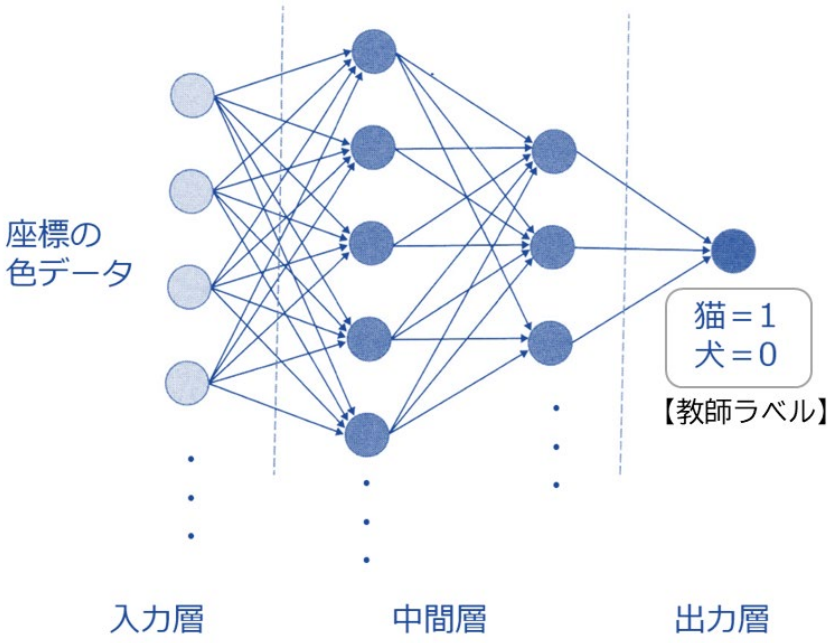
ニューラルネットワークが、犬か猫かをなぜ判定できるかということを簡単に説明したいと思います。

脳は多くのニューロンで構成されており、ニューロン同士が「シナプス」と呼ばれる接続部分でつながり、ネットワーク構造を作っています。脳ではニューロンが他の多数のニューロンから信号を受け取り、その量に応じて他のニューロンに信号を次々に伝達するというメカニズムで情報を処理しています。

人工ニューロンのニューラルネットは、複数のデータを受け取って計算した結果を出力する一種の「関数」だと考えられます。猫の写真のデータは、例えば 100 万画素の CCD カメラで撮影したものであれば、100 万個のデータがあり、色が赤と緑と青(RGB)の 3 色の値で定義されるとすれば、300 万個のデータで構成されていると考えられます。

100 万個の画素データの色情報と位置がわかっていれば、その写真を再現し印刷することができます。たくさんの猫の 300 万個の色データと、たくさんの犬の 300 万個のデータを入力して、猫か犬かをうまく判定できるかのあてはまりのいい関数を作ることが、まさに「教師あり機械学習」です。うまく判定できる関数が作れれば、新しい猫や犬の写真を入力しても、それが猫か犬かということが高い確率で判定できるわけです。もちろん確率的に算出されるので、キツネを入力しても猫と判定するかもしれませんし、タヌキを入力しても犬と判定するかもしれません。また新種の犬を入力した時に猫と判定する可能性もあるわけです。

ニューラルネットは図のような、多層パーセプトロンという構造がよく使われ、ここでは中間層が 2 層ですが、深層学習では 10 層以上あるようなより複雑なネットワークが用いられます。入力層は上記の例でいえば 300 万個のデータがあり、その数値になんらかの係数を掛け算して合計したデータが 1 層目の中間層のデータとなり、さらにそのデータになんらかの係数を掛け算して合計したものが 2 層目の中間層データとなります。そのデータから回答を導くわけですが、適当に選んだ係数では、もちろん正しい結論は導けないわけで、膨大な計算をし、どのような係数の組み合わせにすれば、最も正しく猫か犬かを判定できるかの最適値を導くことで、猫か犬かを判定することができる関数ができあがります。



例題で機械学習を実行しよう（Python を用いた機械学習の実行）

例題では、修学旅行の画像ではなく「どの国の画像なのか」という題材を用意しています。日本、イタリアなどを用意していますが分類名や画像を変更することで、様々な画像を自動で分類することができます。

ニューラルネットワークで画像分類

▾ 学習用のデータ作成

学習用のデータを作成します。
./img/trainにJapanやItalyなどのディレクトリを作成し、画像をアップロードします。
（テスト用のデータは別に残しておいてください。）

ラベルは分類したい数だけ作成します。

```
[2]  # 学習用のデータを作る.
     image_list = []
     label_list = []


     # ./img/train 以下のディレクトリの画像を読み込む。
     for dir in os.listdir("/content/img/train/"):

         dataset_path = "/content/img/train/" + dir
         label = -1

         # Japanはラベル0
         if dir == "USA":
           label = 0
         # Italyはラベル1
         elif dir == "Italy":
           label = 1
         # Chinaはラベル2
         elif dir == "Australia":
           label = 2
         # USAはラベル3
         #elif dir == "USA":
         #  label = 3

         for file in os.listdir(dataset_path):
           # 配列label_listに正解ラベルを追加(Japan:0 Italy:1 China:2 USA:3)
           label_list.append(label)
           filepath = dataset_path + "/" + file
           print(filepath)
           image = Image.open(filepath)
           image = image.resize((500, 500), Image.BICUBIC)
           image = np.asarray(image)
           image_list.append(image / 255.)

     # kerasに渡すためにnumpy配列に変換。
     train_data = np.array(image_list)

     train_label = np.array(label_list)
```

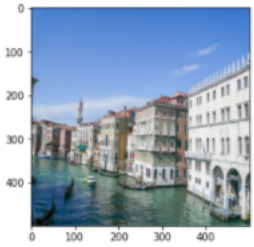
```
/content/img/train/Italy/イタリアーヴェネツィア.jpg
/content/img/train/Italy/イタリアートリノスペルガ聖堂.jpg
/content/img/train/Italy/イタリアーパンテオン神殿.jpg
/content/img/train/Italy/イタリアーヴェネツィア (1).jpg
/content/img/train/Italy/イタリアーピサのドゥオモ広場 (1).jpg
/content/img/train/Italy/イタリアーコロッセオ.jpg
/content/img/train/Italy/イタリアーマテーラ.jpg
/content/img/train/Italy/イタリアートレビの泉.jpg
/content/img/train/Italy/イタリアーアマルフィ海岸 (1).jpg
/content/img/train/Italy/イタリアーアマルフィ海岸.jpg
/content/img/train/Italy/イタリアーヴェナリーア宮殿.jpg
/content/img/train/Italy/イタリアーカテドラーレ大聖堂.jpg
/content/img/train/Italy/イタリアーアルベロベッロのトゥルッリ .jpg
/content/img/train/Italy/イタリアーバチカン市国.jpg
/content/img/train/Italy/イタリアーナポリ歴史地区.jpg
/content/img/train/Italy/イタリアードロミテ.jpg
/content/img/train/Italy/イタリアーピサのドゥオモ広場.jpg
/content/img/train/Italy/イタリアーフィレンツェ歴史地区.jpg
/content/img/train/Italy/イタリアーフォロ・ロマーノ.jpg
/content/img/train/Italy/イタリアーポンペイ.jpg
/content/img/train/Italy/イタリアードロミーティ.jpg
/content/img/train/USA/アメリカ-ホースシューベンド (1).jpg
/content/img/train/USA/アメリカー自由の女神.jpg
/content/img/train/USA/USA-自由の女神.png
/content/img/train/USA/アメリカ-トーマス・ジェファーソン記念館.jpg
/content/img/train/USA/アメリカーヨセミテ国立公園.jpg
/content/img/train/USA/アメリカ-ホースシューベンド.jpg
/content/img/train/Australia/オーストラリアーキュランダ.jpg
/content/img/train/Australia/オーストラリアー　グレートバリアリーフ.jpg
/content/img/train/Australia/オーストラリアーフリーマントル刑務所.jpg
/content/img/train/Australia/オーストリアーウィーン歴史地区.jpg
/content/img/train/Australia/オーストラリアー王立展示館とカールトン庭園.jpg
/content/img/train/Australia/オーストラリアーオペラハウス.jpg
/content/img/train/Australia/オーストラリアーブルーマウンテンズ.jpg
/content/img/train/Australia/オーストリアーハルシュタット.jpg
/content/img/train/Australia/オーストリアーシェーンブルン宮殿.jpg
/content/img/train/Australia/オーストラリアーボートアーサー.jpg
/content/img/train/Australia/オーストラリア-カカドゥ国立公園.jpg
```

```
[3]  import os
     import shutil
     shutil.rmtree('/content/img/train/Italy')
     os.mkdir('/content/img/train/Italy')
```

画像の1つを確認のために表示します。

```
[4]  %matplotlib inline

     import matplotlib.pyplot as plt

     plt.imshow(train_data[3])
```

```
<matplotlib.image.AxesImage at 0x7f7f1b573e50>
```




```
[5]  train_data.shape
```

```
(38, 500, 500, 3)
```

```
[6]  train_label.shape
```

```
(38,)
```

▼ モデルの作成

```
[7]  #train.data.shapeがinput_shape
     model = tf.keras.Sequential([
                                  tf.keras.layers.Conv2D(16, (3, 3),
                                                         input_shape=(500, 500, 3), activation="relu"),
                                  tf.keras.layers.MaxPool2D((2, 2)),
                                  tf.keras.layers.Conv2D(32, (3, 3), activation="relu"),
                                  tf.keras.layers.MaxPool2D((2, 2)),
                                  tf.keras.layers.Flatten(),
                                  tf.keras.layers.Dense(512),
                                  tf.keras.layers.Dense(10, activation="softmax")
     ])
```

```
[8]  model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 498, 498, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 249, 249, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 247, 247, 32)      4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 123, 123, 32)      0
_________________________________________________________________
flatten (Flatten)            (None, 484128)            0
_________________________________________________________________
dense (Dense)                (None, 512)               247874048
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 247,884,266
Trainable params: 247,884,266
Non-trainable params: 0
_________________________________________________________________
```

```
[9]  model.compile(optimizer="adam",
                   loss="sparse_categorical_crossentropy", metrics=["accuracy"])
```

```
[10] #モデルの学習
     np.random.seed(1)
     tf.random.set_seed(2)
     model.fit(train_data, train_label, epochs=20)
```

```
Epoch 1/20
2/2 [==============================] - 12s 4s/step - loss: 13.0444 - accuracy: 0.2100
Epoch 2/20
2/2 [==============================] - 10s 4s/step - loss: 175.3634 - accuracy: 0.3043
Epoch 3/20
2/2 [==============================] - 10s 4s/step - loss: 72.5596 - accuracy: 0.6118
Epoch 4/20
2/2 [==============================] - 10s 4s/step - loss: 37.5139 - accuracy: 0.3043
Epoch 5/20
2/2 [==============================] - 10s 4s/step - loss: 31.3373 - accuracy: 0.3777
Epoch 6/20
2/2 [==============================] - 10s 4s/step - loss: 22.5557 - accuracy: 0.5839
Epoch 7/20
2/2 [==============================] - 10s 4s/step - loss: 2.7882 - accuracy: 0.8459
Epoch 8/20
2/2 [==============================] - 10s 4s/step - loss: 8.2975 - accuracy: 0.7308
Epoch 9/20
2/2 [==============================] - 10s 4s/step - loss: 0.8092 - accuracy: 0.9090
Epoch 10/20
2/2 [==============================] - 10s 4s/step - loss: 0.0019 - accuracy: 1.0000
Epoch 11/20
2/2 [==============================] - 10s 4s/step - loss: 0.2537 - accuracy: 0.9265
Epoch 12/20
2/2 [==============================] - 10s 4s/step - loss: 0.6719 - accuracy: 0.8706
Epoch 13/20
2/2 [==============================] - 10s 4s/step - loss: 0.0548 - accuracy: 0.9545
Epoch 14/20
2/2 [==============================] - 10s 4s/step - loss: 3.8226e-04 - accuracy: 1.0000
Epoch 15/20
2/2 [==============================] - 10s 4s/step - loss: 8.6588e-06 - accuracy: 1.0000
Epoch 16/20
2/2 [==============================] - 13s 5s/step - loss: 1.5522e-06 - accuracy: 1.0000
Epoch 17/20
2/2 [==============================] - 11s 4s/step - loss: 7.8172e-07 - accuracy: 1.0000
Epoch 18/20
2/2 [==============================] - 11s 4s/step - loss: 4.2997e-07 - accuracy: 1.0000
Epoch 19/20
2/2 [==============================] - 10s 4s/step - loss: 2.7286e-07 - accuracy: 1.0000
Epoch 20/20
2/2 [==============================] - 10s 4s/step - loss: 4.5331e-07 - accuracy: 1.0000
<tensorflow.python.keras.callbacks.History at 0x7f7f1bdf4390>
```

```
[11] #学習したモデルをファイルに保存
     model.save("world_heritage.h5")
```

```
[12] #保存したファイルの読み込み
     model_loaded = tf.keras.models.load_model("world_heritage.h5")
```

▼ テスト用データの作成

[13]
```
# テスト用のデータを作る.
image_list = []
label_list = []

# ./data/train 以下のorange,appleディレクトリ以下の画像を読み込む。
for dir in os.listdir("/content/img/test/"):

    dataset_path = "/content/img/test/" + dir
    label = -1

    # Japanはラベル0
    if dir == "USA":
      label = 0
    # Italyはラベル1
    elif dir == "Italy":
      label = 1
    # Chinaはラベル2
    elif dir == "Australia":
      label = 2
    # USAはラベル3
    #elif dir == "USA":
    #  label = 3

    for file in os.listdir(dataset_path):
      # 配列label_listに正解ラベルを追加(China:0 Italy:1 Japan:2 USA:3)
      label_list.append(label)
      filepath = dataset_path + "/" + file
      print(filepath)
      image = Image.open(filepath)
      image = image.resize((500, 500), Image.BICUBIC)
      image = np.asarray(image)
      image_list.append(image / 255.)

# kerasに渡すためにnumpy配列に変換。
test_data = np.array(image_list)

test_label = np.array(label_list)
```

```
/content/img/test/Italy/USA-自由の女神 - コピー.png
/content/img/test/Italy/イタリアーヴェネツィア (1).jpg
/content/img/test/Italy/オーストラリアー　グレートバリアリーフ.jpg
/content/img/test/Italy/エジプトーギザのピラミッド.jpg
/content/img/test/Italy/オーストリアーシェーンブルン宮殿.jpg
/content/img/test/Italy/イタリアーピサのドゥオモ広場.jpg
```

[14]
```
#正解率
pred = model_loaded.predict_classes(test_data)
(pred == test_label).sum() / len(test_label)
```

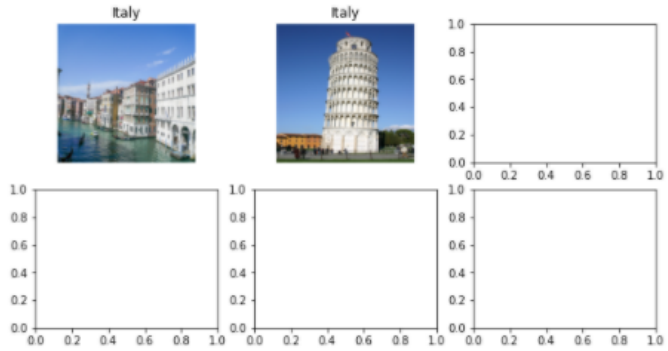
```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be rer
  warnings.warn('`model.predict_classes()` is deprecated and '
0.3333333333333333
```

▼ 実行結果

```
[15] label_names = ["USA", "Italy","Australia"]

     correct_data = test_data[pred == test_label]
     correct_label = pred[pred == test_label]

     print(correct_label.shape)
```

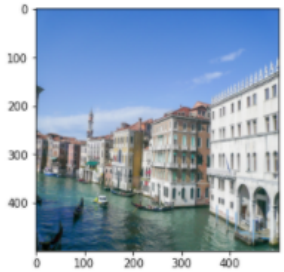
```
(2,)
```

```
[19] #正解した画像を表示
     fig, axes = plt.subplots(2, 3, figsize=(10, 5))
     for i in range(2):
       ax = axes[i // 3][i % 3]
       ax.set_title(label_names[correct_label[i]])
       ax.axis("off")
       ax.imshow(correct_data[i])
```




```
[20] plt.imshow(correct_data[0])
```

```
<matplotlib.image.AxesImage at 0x7f7f12e5c550>
```



リカレントネットワーク

リカレントネットワークは、ニューラルネットワークの一種です。前にモデルに取り入れた情報を、次の予測に使うことができる回帰(リカレント)構造を持つため、時系列データや、言語データといった連続性を持ったデータの予測に使われます。連続性を商品のレビューやコールセンターの音声データから、顧客の満足度を測る感情分析も、こちらの手法が使われています。これは、単語の意味を、前後の文脈を考慮して予測することができるからです。

ディープラーニング

ディープラーニングは、ニューラルネットワークの中間層を増やしたものです。中間層を増やすことによって、より複雑な問題を解いたり、予測精度を上げたりすることができます。その一方、モデルが複雑すぎて、どのように予測が導かれたのか人間には理解できないため、ブラックボックスモデルと呼ばれることもあります。
2016 年に囲碁の世界チャンピオンを倒した、AlphaGo もこちらの手法を使って開発されました。

ナイーブベイズ法

ナイーブベイズ法は、ベイズの定理を用いてデータを分類する規則を見つける手法です。

古くから使われている手法であり、特に自然言語処理の分野で多く使われています。例えば、ニュース記事を政治・経済・スポーツなどのカテゴリに分けたり、電子メールを迷惑メールとそうでないものに分ける際に、ナイーブベイズ法は高い性能を発揮します。

Case　記事を自動で分類してみよう

晴斗さんはスポーツニュースサイトの運営を行なっています。サイトでは、クラスメイトに協力してもらってサッカー・野球のどのような点が好きかを記事として書いてもらっています。

これまでは晴斗さんが自ら、記事を「野球」と「サッカー」のどちらかのカテゴリに分類していました。
しかし、これからはナイーブベイズ法を用いて、新しい記事に含まれる単語の種類から、
その記事を自動的に分類する仕組みを作ることにしました。
そこで晴斗さんは、さっそく過去の記事とカテゴリを調べました。

まず、サイト全体の記事のうち、各カテゴリの記事の割合を計算すると、
野球が 6 割、サッカーが 4 割でした。次に、サイト内の記事でよく使われる単語を
リストアップし、各単語が記事に出現する割合をカテゴリ別に計算すると、
次のようになりました。

ここでは問題を簡単にするため、単語の数を全部で 3 種類にしています。

単語	野球カテゴリ	サッカーカテゴリ
試合	0.6	0.5
シュート	0.1	0.6
ドーム	0.2	0.1

このとき、「試合、結果、シュート」の 3 つの単語で構成された新しい記事が、「野球」もしくは「サッカー」のどちらのカテゴリに近いのか、ナイーブベイズ法を使って分類しましょう。

方針として、分類対象の単語の種類から、「野球」「サッカー」それぞれのカテゴリに分類される確率を求め、その確率が大きい方をその記事のカテゴリとします。

アルゴリズムと解法

ナイーブベイズ法のアルゴリズムは「ベイズの定理」とそれに必要な「確率」で形成されています。以下、分類対象の記事を D、カテゴリを C_i と表します。i はカテゴリの種類を表し、C_1 を野球カテゴリ、C_2 をサッカーカテゴリとします。

このとき、ベイズの定理を使うと、以下の式が成り立ちます。

$$P(D) = \frac{P(C_i)P(C_i)}{P(D)}$$

ここで、$P(C_i)$はC_iカテゴリが発生する確率であり、事前確率とよばれます。今回は、各カテゴリの事前確率は次の値になります。

$$P(C_1) = 0.6\,, P(C_2) = 0.4$$

左辺の$P(C_i|D)$が今回求めたい値です。記事Dが生成されたという条件の下でC_iカテゴリが発生する条件付き確率であることから、事後確率とよばれます。この事後確率を「野球」「サッカー」カテゴリそれぞれで求め、大きい方を記事Dのカテゴリとします。

$P(D|C_i)$はC_iカテゴリにおいて記事Dが生成される確率です。
これは、データの特徴（今回は記事に出現する単語の種類）が、そのカテゴリに属する尤も（もっとも）らしさを表しているので尤度（ゆうど）と呼ばれます。この尤度を厳密に計算するのは難しく、ナイーブベイズ法では、記事の生成モデルにある仮定をおきます。それは、「C_iカテゴリにおいて、各単語は独立である」というものです。つまり、「ある単語の出現確率は、他の単語の出現確率の影響を受けない」とする仮定です。

本来は、サッカーの記事で「ゴール」という単語が含まれていれば、「シュート」「勝利」といった関連する単語もその記事に含まれていてもおかしくありません。すなわち、「ゴール」という単語の出現が、他の単語の出現確率に影響を与えていることになります。これは自然な考え方ですが、この考えが尤度の計算を複雑にするので、ナイーブベイズ法では各単語の出現確率は互いに影響を与えないと仮定します。
すると、今回の尤度は以下のように変形できます。

$$P(D|C_i) = P(C_i) \times P(C_i) \times P(C_i)$$

すると、各カテゴリの尤度は以下のようになります。

$$P(D|C_1) = P(C_1) \times P(C_1) \times P(C_1) = 0.6 \times 0.1 \times 0.8 = 0.048$$

$$P(D|C_2) = P(C_2) \times P(C_2) \times P(C_2) = 0.5 \times 0.6 \times 0.9 = 0.27$$

$P(D)$は記事Dが発生する確率ですが、今回は計算しません。なぜなら、各カテゴリの事後確率を求めなくても、その大小関係さえわかればクラス分類が可能だからです。

ベイズの定理の右辺を見ると、分母の$P(D)$はカテゴリC_iを変えても変化しません。つまり、分子部分だけを計算すれば各カテゴリの事後確率を比較できるのです。

以上から、各カテゴリの事後確率の、分子部分は、

$$P(D|C_1)P(C_1) = 0.048 \times 0.6 = 0.0288$$

$$P(D|C_2)P(C_2) = 0.27 \times 0.4 = 0.108$$

となり、記事Dはサッカーカテゴリと分類されました。

ナイーブベイズ法を用いて記事のカテゴリ分類ができました。問題例では 2 つのカテゴリで分類しましたが、もちろんもっと多くのカテゴリや単語数でも同様に考えることが可能です。

尤度の箇所で説明した通り、ナイーブベイズ法では条件付き独立という仮定を置いています。ナイーブが「単純な」を意味する通り、この仮定を置くことにより計算がシンプルになりました。これはあくまで仮定であり、本来は独立ではありません。しかし、それでも高い性能を発揮することが多く、古典的な手法ながら現在でも使われています。

例題で機械学習を実行しよう（Python を用いた機械学習の実行）

※例題のデータセットでは、ニュース記事ではなく、クラスメイトに「好きなスポーツ」について記述したテキストを用意しました。

テキストのナイーブベイズ分類器

```
#必要なライブラリのインポートを行う TF-IDFを用いたテキストの特徴量化を用いて教師あり学習を実行する
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import  fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
```

学習用のデータ作成

```
[21] # 学習用のデータを作成.
     image_list = []
     label_list = []


     # ./img/train 以下のディレクトリのテキスト文書を読み込む
     for dir in os.listdir("/content/img/train/"):

         dataset_path = "/content/img/train/" + dir
         label = -1
```

```
[10] train = favorite_text(subset='train')
     test = favorite_text(subset='test')
```

```
[ ]  学習用のデータを作成します。
     ./img/trainにSoccerやBaseballなどのディレクトリを作成し、テキストをアップロードします。
     （テスト用のデータは別に残しておいてください。）

     ラベルは分類したい数だけ作成します。
```

```
[11] print("Train Count =", len(train.data))
     print("TestCount =", len(test.data))
```

ナイーブベイズは「分類」を行いますので結果は混同行列で確認すると良いでしょう。また、今回は文章を特徴量化する方法として提供している Python サンプルコードでは TF-IDF というライブラリを使用しています。文章（テキスト）は非構造化データであり、そのままでは表形式のデータではないため、表形式（行列）として扱えるよう変換が必要です。その変換を行い、さらに扱いやすくしているのが TF-IDF です。このライブラリが何を行っているかについても、調べてみると良いでしょう。

K 近傍法

K 近傍法は、分類に使われる手法の一つで、与えられた学習データと入力データとの距離を計算し、距離の近い順に探し出した K 個の学習データと、入力データとの多数決で得られた結果を、分類結果とする比較的シンプルなアルゴリズムです。このアルゴリズムを利用し、予測したい入力データ X について、既に値が明らかになっている学習データの中から X と似ているデータを見つけ出し、そのデータの参考にすることで値を予測します。

Case　**アニメの好みが似ている人を探して、その人がおすすめしてくれるアニメを視聴してみよう。**

翔真くんはアニメを見るのがとても好きです。

その中でも、とりわけ宇宙飛行士を夢見る小学生が主人公の宇宙×バトルのアニメ、心優しい宇宙人と人間が交流を深めていく宇宙×友情がテーマになっているものをよく観ます。

ただ、それらを扱ったアニメは数が少なく、全て観てしまいましたそこで翔真さんは、「扱うテーマは違っても、似たようなテイストのアニメがわかればいいのに」と考えるようになりました。

「そういえば、ジャンルごとにタグ付けされているアニメの評論サイトがあったな。」

「感想を書いているユーザー別に、これまで見てきたアニメのタグのカウントや、今の一押しなんかも発表されていたな。」

「宇宙、バトル、友情みたいなタグの総数が多い人は、僕とアニメの好みが似ているかも！
その人がおすすめしているアニメは、私も楽しめるかもしれない！」

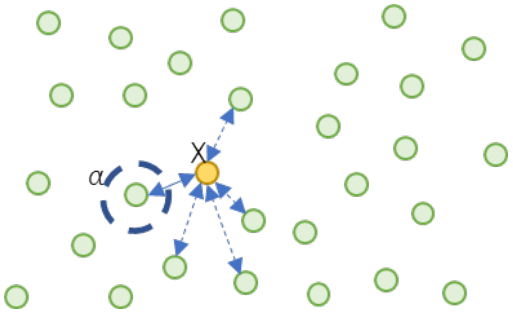
翔真くんはふと、学校で習った授業のことを思い出しました。

「そういえば、似ている人」を特定して、似ていないところを探してみる、

というのを習ったような・・・ 考えてみようかな？

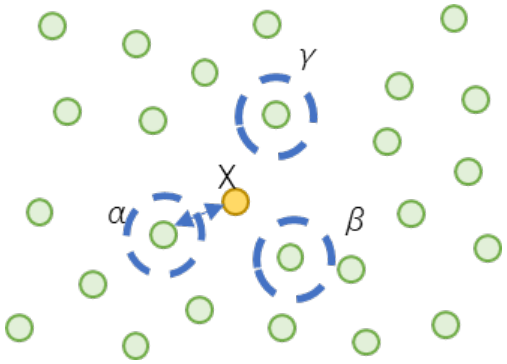
ここでは「似ている人」を同じ好みのグループに所属しているとして考えてみましょう。

	アニメのジャンルの視聴回数				
	宇宙	バトル	友情	魔法	所属グループ No
翔真くん	10	10	10	0	1
美織さん	5	1	10	20	2
雄大くん	10	11	12	1	3
芹那さん	5	20	20	0	1



距離の近いデータ

K 近傍法では、似ているデータを見つけ出すために、SVM と同様にユークリッド距離を使うのが一般的です。データ間の距離（二点間の距離）は三平方の定理で計算することができます。いま、求めたいデータ X に一番近いデータは α となっています。



K=3 のとき

K 近傍法の K は参考にする値をいくつにするかを示します。K の値については、二値分類の場合、K を奇数にすることで多数決の結果がどちらかに決まるようにすることができます。上記では K＝3 とし、似ているデータ α～γ の 3 つを点線で囲っています。X の値は 3 点のデータ α～γ の値の平均値をとります。しかし、この時 K の値を大きくしすぎないように注意が必要です。K を大きくしすぎると距離が近くないデータを参考にする可能性があり、そうなると予測の精度が落ちてしまいます。これを防ぐために、加重平均を利用する方法があります。

<<加重平均を用いた予測>>

K 近傍法で、平均をとるということは、距離の近い 3 点をすべて同じくらい参考にするということです。しかし、現実的にはよく似ているものを 1 番参考にし、2 番目に似ているものをその次に参考にする･･･とした方がより良い結果を得ることができる可能性があります。

K の値が増えると参照する値が増えるということで、X からの距離も離れて行くことになる場合もあります。離れすぎてしまうと、あまり似ていないので参考にする程度を考える必要が出てきます。

これを解決するために、単純に平均をとるのではなく、距離の近さを加味し重みづけをした平均「加重平均」をとります。距離の近さを考慮しますので、距離が近い(値が小さい)方は大きな値に、また距離が遠い(値が大きい)方を小さな値にするような重みづけを行います。簡単な重みづけにするなら、重みを距離の逆数(1 を距離で割る)にすることで可能になります。

例題で機械学習を実行しよう(Python を用いた機械学習の実行)

▾ k近傍法

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
```

▾ 前処理

```
iris = datasets.load_iris()
df = pd.DataFrame(
    iris.data,
    columns = iris.feature_names
)
df["label"] = iris.target

df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	label
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
[ ] X = iris.data     # 説明変数
    Y = iris.target     # 目的変数
```

学習データとテストデータを7:3で分割します。

```
[ ] from sklearn.model_selection import train_test_split

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3)
```

```
[ ] from sklearn.neighbors import KNeighborsClassifier

    knn = KNeighborsClassifier(n_neighbors=6)
    knn.fit(X_train, Y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=6, p=2,
                     weights='uniform')
```

```
[ ] # 予測実行
    y_pred = knn.predict(X_test)
    y_pred
```

```
array([2, 2, 0, 1, 0, 2, 0, 0, 2, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 2, 1, 2, 0, 0, 2, 2, 2, 2, 0, 1, 1, 0, 0, 2, 0, 2,
       1])
```

予測精度を確認します。

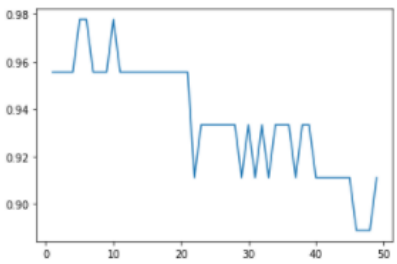
```
[ ] from sklearn import metrics

    metrics.accuracy_score(Y_test, y_pred)
```

```
0.9777777777777777
```

kの値によって予測精度が変わるので、kを変化させながら予測精度を出力するグラフを参考までに作成します。

```
k_range = []
accuracy = []

for k in range(1, 50):
    knn = KNeighborsClassifier(n_neighbors=k) # インスタンス生成
    knn.fit(X_train, Y_train)                 # モデル作成実行
    y_pred = knn.predict(X_test)              # 予測実行
    accuracy.append(metrics.accuracy_score(Y_test, y_pred)) # 精度格納
    k_range.append(k)

plt.plot(k_range, accuracy)
```

```
[<matplotlib.lines.Line2D at 0x7f048330d410>]
```



結果の解釈

予測の精度が 0.97 と高い結果となりました。K の数を増やしていくと、下がっていくことがわかります。過学習を考慮するためには、K をいくつに設定すると良いでしょうか。K の値が少ないと、学習精度は高いですが過学習とならないかが気になります。一方で、K の値を高く設定すると精度は下がります。精度を確認しながら、学習データと評価データの分割比を変更したり、K の値をチューニングしたり、説明変数の取捨選択を行って過学習を防ぐようチューニングしましょう。

アノテーション

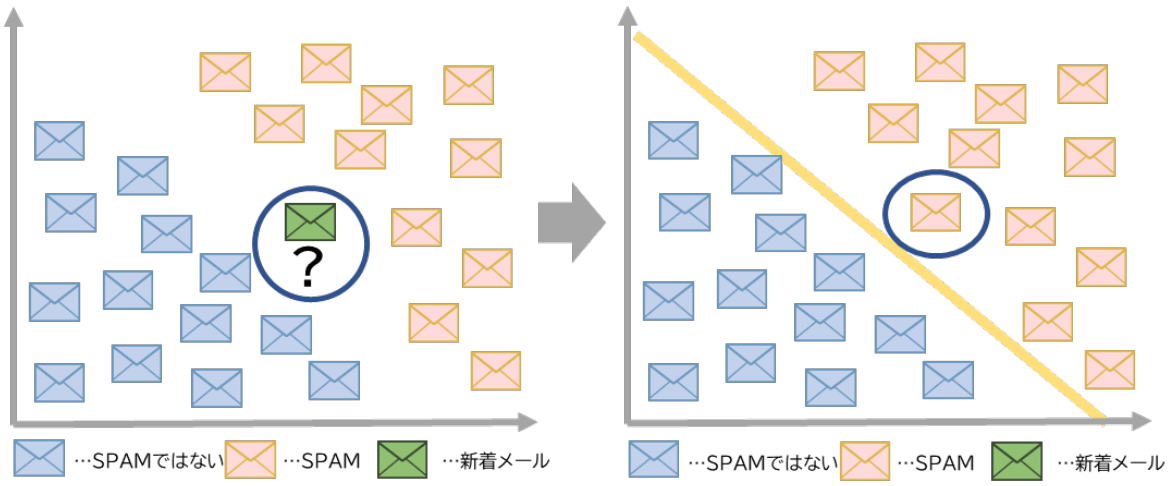
データに正解ラベルを貼る「アノテーション」

アノテーションとは、テキストや画像、音声など様々な形式のデータに対して、データに意味を持たせるタグと呼ばれる情報を付与することです。

機械学習の教師あり学習においては、学習のもととなる正解データ（いわゆる「教師データ」）を高い精度で作ることがとても大切です。

身近なアノテーションの事例：迷惑メールの判定

身近なアノテーションの事例では、受信メールボックスにおける迷惑メールの自動振り分けが挙げられます。受信の度に、メールを迷惑メールかそうでないかを振り分けているこの作業もアノテーションの一種です。



アノテーションの種類

アノテーションは、用途やデータ形式によって作業内容が異なります。代表的なものを 3 つ、以下に説明します。

・テキスト
・画像
・音声

意味的(セマンティック)アノテーション

フリーテキストと呼ばれる文章などのテキストデータに対して、事前に定義した「人」「モノ」「企業名」といったカテゴリを割り当てる作業です。文章中の文や段落を、割り当てられたカテゴリ情報を使用して、トピック毎に振り分けることが可能となります。

ニュースサイトなどで「国際」「政治」「娯楽」「スポーツ」といったように分類されているのも、この方法を活用した事例のひとつです。

画像・映像アノテーション

画像情報に対するアノテーションでは、①写っているものを検出して関連する言葉のタグ(猫、植物など)をつける**物体検出**と、②画像の中で特定の範囲を指定してタグ付けをする**領域抽出**、③画像そのものに対してタグ付けを行う**画像分類**とがあります。

音声アノテーション

大量の音声データを書き起こしテキスト化するところからはじまります。

例えば、あるサービスのコールセンターでの顧客との会話のやりとりなどの録音データを使用して、「商品名」「メーカー名」「機種名」などの情報の抽出のほか、接続詞や感動詞など単語一つ一つに意味をタグ付けしていきます。

過学習と次元の呪い（及びその対処法）

機械学習を行う中で、過学習や次元の呪いとよばれる問題に出会うことがあります。過学習（あるいは overtraining, 過適合（overfitting））とは学習データに対する適合は良くなる一方で、テストデータに対しての予測精度は落ちてしまうことを言います。次元の呪いとは、データの次元数が大きい場合、モデル作成に必要な計算量が増えて効率的に機械学習が行えなくなることを言います。

過学習とは

例えば、ある大学を目指す生徒の集団に関して、勉強時間と日々の小テストの点数が軸にとってあるプロットを考えます。そのプロット上に、大学受験で合格したか否かで分離する線を引くとしましょう。勉強時間と小テストの点数がある一定以上の生徒たちとそれ以外の生徒たちで分類されることは予想できます。その分離条件を元にすると、次年の生徒たちに関して合否が予測できます。しかし、そのデータには勉強時間や小テストの点数が、一定以上であることを満たさなくとも合格する生徒が数人いるとしましょう。

勉強時間が少なく小テストの点数が低くても本番の問題と相性が良くて受かってしまう生徒などを想像してください。そのような数人に関しても完全に分離ができるように複雑な線を引いたところで、その線にはあまり意味がなさそうです。その分離線を用いて次年の生徒の合否を予測しても精度が低いことには納得がいくでしょう。

このように、ある学習データ（今年の大学受験合否と勉強時間、小テストの点数に関するデータ）に対してあまりにも厳密なモデル（分離の線）を考えると、かえって違うデータに対する予測（次年の生徒の合否）精度が落ちてしまうことを過学習といい、機械学習ではたびたび問題になります。それを防ぐためには、データへの当てはまりの良さとモデルの簡潔さのバランスを取ることが良いとされています。統計解析においては、そのバランスを見るための様々な目安（情報量基準）が知られています。また、学習データに関して学習の制限を設ける正則化というような操作も有効だと知られています。

次元の呪いとは

ここで、冷蔵庫にあるものの中からおいしい料理を作ることを考えます。冷蔵庫に麻婆豆腐の素とお豆腐しか入っていない場合、考えられるレシピは麻婆豆腐か冷ややっこ、湯豆腐など頑張っても 3 品ほどでしょう。ただ、カレー粉と人参、ジャガイモ、玉ねぎ、お肉、糸こんにゃくなどが入っているとします。するとカレー、肉じゃが、豚汁、カレー炒め、ポトフ、ちょっと頑張って野菜のかき揚げなども考えられます。他に、名がなくともおいしい料理を作ることができそうです。このように「おいしい料理を作る」という目標に対して、冷蔵庫の中にあるものを組み合わせて対処する場合、何か材料が増えただけでそのバリエーションはぐっと増えます。ここで、最高においしい料理を作成して友人をもてなそう、と考えるとします。

そのメニューを考えるとき、材料が少ない場合は材料ごとの組み合わせも少なく、満足いくまで料理を作って食べ比べることができます。しかし、材料が多いと色々な組み合わせで料理を作り、味見をし、ということを繰り返すのには多大な時間と労力を要します。これはデータ解析にも当てはめることができます。

データ解析で売り上げを気温や湿度、客層などによってあらわすモデルを作るとします。良いモデルを作るためには必要な変数を選択してそれぞれに重み付けなどを行い、売り上げをよく表現する必要があります。モデルを作成するために用いるデータに関して次元(変数の個数)があまりにも多いと、変数同士の組み合わせが指数関数的に増加してしまいます。その結果、恐ろしいほど計算量が増え、手持ちのデータだと十分な学習結果が得られないことがあります。

このようにデータの次元数が多すぎることによって機械学習の効率が落ちたり、満足のいく結果が得られないことを「次元の呪い」といいます。これを避けるには、複数の変数を一つの特徴量にまとめたり(特徴量作成)、有効な特徴量の組み合わせを絞ったり(特徴量選択)することが有効だと言われています。