教師あり学習（回帰）の基礎

- 1.1 教師あり学習とは
  - 1.1.1 機械学習の種類
  - 1.1.2 scikit-learnを用いた機械学習
- 1.2 線形回帰
  - 1.2.1 線形回帰とは
  - 1.2.2 決定係数
  - 1.2.3 線形単回帰
  - 1.2.4 線形重回帰
- 1.3 添削問題

1.1 教師あり学習（回帰）とは

1.1.1 機械学習の種類

機械学習は主に3つの分野に分かれます。

1つは **教師あり学習** と呼ばれるものです。 蓄積されたデータを元に機械が新しいデータや未来のデータの予測、あるいは分類を行うことを指します。 株価の予測や画像識別などが当てはまります。

教師なし学習 と呼ばれる分野も存在します。 蓄積されたデータの構造や関係性を機械が見出すことを指します。 小売店の顧客の傾向やGoogleの画像認識などで用いられています。

最後に **強化学習** です。 学習形態は教師なし学習に近いのですが、報酬や目標などを設定することで学習時に利益の最大化を図るように学習をする手法です。 囲碁などの対戦型AIとして用いられていることが多いです。

このなかで、教師あり学習は2つの手法に大別されます。

これから学ぶのは **回帰** です。 既存のデータから関係性を読み取り、その関係性を元にデータの予測を行う手法です。 予測される値は株価や宝石の時価などの連続値になります。

もう一つの分野である **分類** については教師あり学習（分類）という別講座を参照ください。 こちらもデータの予測を行うのが主目的ですが、予測される値はデータのカテゴリーであり、離散値となります。

問題

- 次のうち教師あり学習の回帰について説明しているものを選んでください。

1. データの構造や関係性を機械が学習によって見出す手法。
2. 株価の時間変化や家屋の敷地や装飾などによる価格の変化などの連続的関係性を見出しデータを予測する手法。
3. データの特徴を学習し、データをカテゴライズする手法。
4. 機械に具体的な目標を設定し、利益が最大となるように学習させる手法。

ヒント

- 教師あり学習はデータの予測が主目的です。
- 回帰は関係性が連続的なのが特徴です。

解答

株価の時間変化や家屋の敷地や装飾などによる価格の変化などの連続的関係性を見出しデータを予測する手法。

1.1.2 scikit-learnを用いた機械学習

また、機械学習のためのモジュールであるscikit-learnを利用します。

scikit-learnを用いた機械学習の流れをサンプルコードとともに掲載します。

```
# 必要なモジュールのインポートします。
import request
from sklearn.linear_model import LinearRegression

# 次に学習させたいデータの読み込みを行います。詳しいコードはこの問題にあるコードを参照ください。
# 以下のようにtrain_X, test_X, train_y, test_yという4つのファイルに分けてデータがロードします。
train_X, test_X, train_y, test_y = (データの情報)

# 学習器の構築を行います。
# 学習器とは、学習モデル(学習方法)に沿って学習を行うように設計されたオブジェクトのことです。
# scikit-learnのLinearRegressionが学習して予測データを返してくれるのです。
# このLinearRegressionの詳細は次以降のセッションで扱います。
model = LinearRegression()

# 教師データ(学習を行うための既存データ)を用いて学習器に学習させます。
model.fit(train_X, train_y)

# 教師データとは別に用意したテスト用データを用いて学習器に予測させます。
pred_y = model.predict(test_X)

# 学習器の性能を確認するため決定係数という評価値を算出します。
score = model.score(test_X, test_y)
```

練習問題では簡単な学習の流れを確認してみましょう。

問題

- 次のプログラムはデータを学習するために必要なコードが抜けているために実行するとエラーになります。
- 必要なコードを埋めてプログラムを完成させてください。

In [ ]:

```
# scikit-learnのLinearRegressionというモデルをインポートします。詳細は1.2で説明します
from sklearn.linear_model import LinearRegression

# scikit-learnに標準で搭載されている、ボストン市の住宅価格のデータセットをインポートします
from sklearn.datasets import load_boston

# scikit-learnに搭載されているデータセットを学習用と予測結果照合用に分けるツールをインポートします
from sklearn.model_selection import train_test_split


# データの読み込みです
data = load_boston()

# データを教師用とテスト用に分けます
train_X, test_X, train_y, test_y = train_test_split(
    data.data, data.target, random_state=42)

# 学習器の構築です
model = LinearRegression()

# 教師データを用いて学習器に学習させてください


# テスト用データを用いて学習結果をpred_yに格納してください


# 予測結果を出力します
print(pred_y)
```

ヒント

- 出力するのは予測した値です。評価値ではありません。

解答例

In [ ]:

```
# scikit-learnのLinearRegressionというモデルをインポートします。詳細は1.2で説明します
from sklearn.linear_model import LinearRegression

# scikit-learnに標準で搭載されている、ボストン市の住宅価格のデータセットをインポートします
from sklearn.datasets import load_boston

# scikit-learnに搭載されているデータセットを学習用と予測結果照合用に分けるツールをインポートします
from sklearn.model_selection import train_test_split


# データの読み込みです
data = load_boston()

# データを教師用とテスト用に分けます
train_X, test_X, train_y, test_y = train_test_split(
    data.data, data.target, random_state=42)

# 学習器の構築です
model = LinearRegression()

# 教師データを用いて学習器に学習させてください
model.fit(train_X, train_y)

# テスト用データを用いて学習結果をpred_yに格納してください
pred_y = model.predict(test_X)

# 予測結果を出力します
print(pred_y)
```

1.2 線形回帰

1.2.1 線形回帰とは

回帰分析とは、予測したいデータを、すでにわかっているデータとの関係性を元に推定するアプローチのことです。最終的には **数値を予測する時に「回帰」** という呼び方をします。

例えば毎分4L水が溜まるタンクがあるとします。5分後には水の量はどれくらいになっているでしょうか。おそらく20Lと予想がつくと思います。なぜならば$\mathrm{V(L)}$を水の容量、$\mathrm{t(m)}$を時間とした時に

$$\mathrm{V} = 4\mathrm{t}$$

というモデルを我々の頭の中で組んでいるからです。ここで線形（=グラフの形が直線）であるというのはtが1次（2乗などになっていない）ということを表します。（厳密には、機械学習における線形回帰では、tが2次以上のもの（=グラフの形が曲線）もその範囲として扱われます。これは、重みを調整する際の計算式を紐解くと1次式（=線形/グラフの形が直線）となるためです。）

線形回帰では予測に用いる **データの係数（ここでは4）を見ることで予測したいデータに対するそのデータの寄与の大きさを見る** ことができます。

データの寄与の大きさを見るもう一つの例を挙げてみます。

あるレストランでは、アンケートが実施されていて、総合評価を100点満点、食べ物のおいしさを50点満点、接客の良さを50点満点でつけるとします。

店長が、総合評価の点数を良くするためには、食べ物のおいしさと接客の良さ、どちらかに力を入れたら良いか悩んでいるとしましょう。

また、利用客は、食べ物のおいしさの点数の点が高いと総合評価の点数も90点ぐらいをつけていて、接客の良さの点数が高くても食べ物のおいしさの点数の点が低いと、総合評価が30点くらいになるような点数のつけ方をしていたとします。

この時、総合評価の点数をT(点)、食べ物のおいしさの点数を$\mathrm{P_1}$(点)、接客の良さの点数を$\mathrm{P_2}$(点)とすると、

$$\mathrm{T} = 2\mathrm{P_1} + 0.5\mathrm{P_2}$$

といったモデルを組むことができます。

このモデルの$\mathrm{P_1}$、$\mathrm{P_2}$の係数を見ると、食べ物のおいしさの点数$\mathrm{P_1}$の寄与は大きく、接客の良さの点数$\mathrm{P_2}$の寄与は小さいことがわかります。

そのため、店長は、接客の良さよりも食べ物のおいしさに力を入れるべきだ、と考えることができるのです。

<img src="https://aidemyexcontentspic.blob.core.windows.net/contents-pic/5010_regression/linear.png (https://aidemyexcontentspic.blob.core.windows.net/contents-pic/5010_regression/linear.png)", width="600">

問題

- 線形回帰に対する次の説明のうち正しいのはどれでしょうか。

- 線形回帰は予測するデータの値（数値）を関数を用いて予測する方法である。
- 線形回帰は予測するデータの値（カテゴリ）を関数を用いて予測する方法である。
- 線形回帰は回帰分析において唯一の方法である。
- 線形回帰はデータによって分析したい事象を完全に分析できる。

ヒント

- 線形回帰によるアプローチは事象を説明するための一つの方法です。
- 線形は予測に用いるデータの一次関数(グラフにすると直線になります)の総和であることからその名前があります。

解答

線形回帰は予測するデータの値（数値）を関数を用いて予測する方法である。

1.2.2 決定係数

決定係数 とは、線形回帰で予測したデータと実際のデータが、どのくらい一致しているかを示す指標です。また、各データの係数(寄与の大きさ)をどれくらい信用していいかも示しています。

先ほど(1.2.1)のレストランの総合評価の点数の例で説明します。

アンケートの点数が、

総合評価：80点　食べ物のおいしさ：40点　接客の良さ：20点

だったとします。また、線形回帰で求めた関数に、食べ物のおいしさ：40点　接客の良さ：20点の情報を与えると、予測された総合評価の点数が50点だったとします。このとき、**予測した点数と実際の点数に大きく差があるため、決定係数の値は0に近づきます。**

逆に、予測された総合評価の点数が78点だった場合、**予測した点数と実際の点数の差が非常に小さいため、決定係数の値も1に近づきます。**

ここでは決定係数を求めるための詳しい数式は紹介しませんが、 **決定係数は0から1までの数字** を取り、 **値が大きければ大きいほど関数の精度が良い** ことを表しています。おおよそ0.8以上の数値であればその関数の精度は良いと見ることができます。 しかし、**0.8以下の数値であってもその関数が無駄なわけではありません。**

店長の悩みは、食べ物のおいしさと接客の良さ、どちらに力を入れるべきかでした。決して、総合評価の点数を見事的中させたいわけではありません。

決定係数がある程度の大きさ(人によって基準は変わるが、0.4以上くらい)であれば、データの寄与の大きさはある程度信頼できます。つまり、接客の良さよりも食べ物のおいしさに力を入れるべきだと考えても問題ありません。

問題

- 決定係数に対する次の説明のうち正しいのはどれでしょうか。

- 決定係数が小さいほど関数の精度が良い。
- 決定係数は関数が予測するデータと実際のデータにどれくらい差があるかを示している。
- 決定係数が小さい場合入力するデータの数を増やせばよい。
- 決定係数は機械学習において自分で定めることができる。

ヒント

- 決定係数は関数の精度、データの寄与の大きさの信頼度を示します。
- モデルが変わると決定係数も変わります。

解答

決定係数は関数が予測するデータと実際のデータにどれくらい差があるかを示している。

1.2.3 線形単回帰

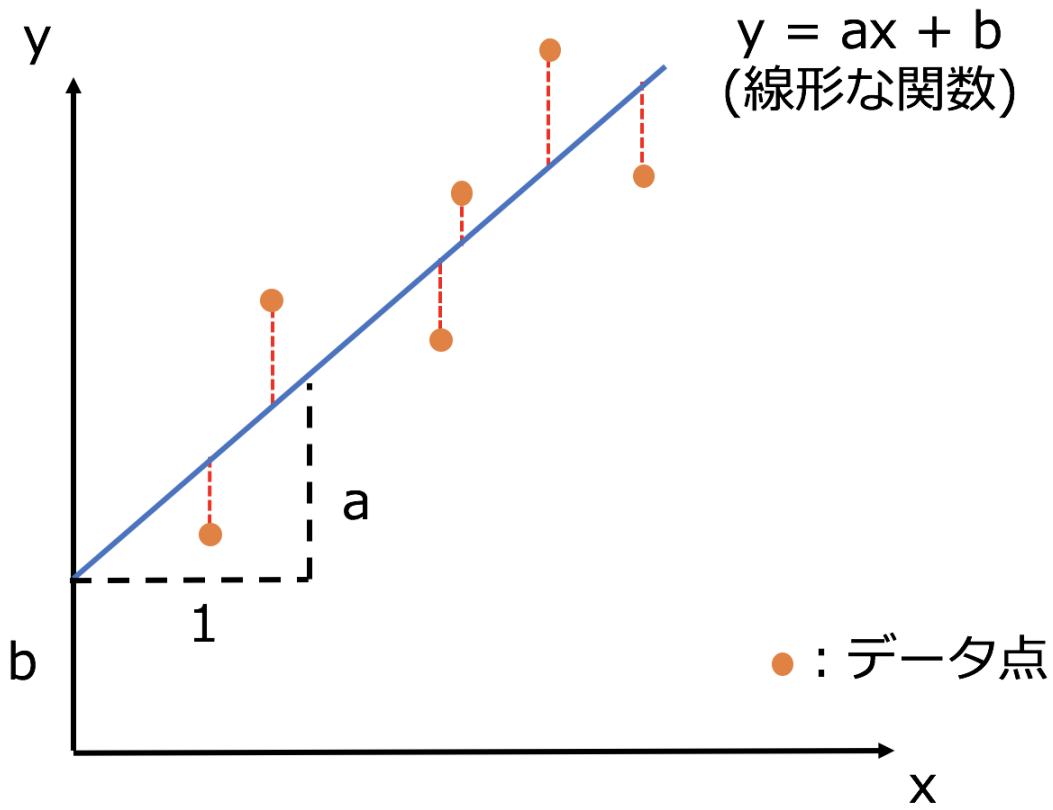
線形単回帰 とは、1つの予測したいデータ(ex. 水の量)を **1つのデータ(ex. 時間)から求める回帰分析** です。データの関係性を調べるときに使うことが多く、予測を行うときに用いられることは稀です。

ここでは予測したいデータをy、予測に用いるデータをxとして、

$$y = ax + b$$

という関係があると仮定して、aとbを推定します。このチャプターの「線形回帰とは」の水の量のたとえと同じ形の数式なのがわかります(a=4、b=0)。

aとbの推定には様々な方法がありますが、今回は **最小二乗法** と呼ばれる方法を用いましょう。 実際のyの値と推定するy(= ax + b)の値の差の **二乗の総和** が最小になるようにaとbを定める方法です。下図で言うと、オレンジのデータ点からの距離の総和が最小になるようにa、bを決めます。 このようにしてすでにあるデータに対して一番近い直線を引き、その直線から今後のデータを推測します。



なお、ここで誤差を二乗するのは何故でしょうか？それは **二乗することによって、正負の相違による誤差のキャンセルがされないようにする** ためです。例えば誤差が+2と-2のものを単純に足し合わせると、値が0になって誤差がキャンセルされてしまいます。

さて、実際に回帰分析を行うには、scikit-learnのlinear_modelモジュール内にあるLinearRegressionというモデルを使うのが便利でしょう。

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# 1.1.2では既存のデータセットを用いましたが、ここでは回帰的なデータの生成をします。
X, y = make_regression(n_samples=100, n_features=1, n_targets=1, noise=5.0, random_st
ate=42)

train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42)

model = LinearRegression()

model.fit(train_X, train_y)

# 決定係数の出力です
print(model.score(test_X, test_y))
```

問題

- 線形単回帰を実行してみます。
- test_X, test_yに対する決定係数を出力してください。
- 出力はprint関数を用いてください。

In [ ]:

```
# 必要なモジュールのインポート
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# データの生成
X, y = make_regression(n_samples=100, n_features=1, n_targets=1, noise=5.0, random_state=42
)
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42)

# 以下にコードを記述してください。




# test_X, test_yに対する決定係数を出力してください
```

ヒント

- 決定係数は model.score(data, target) で算出できます。
- モデル名は LinearRegression() です。

解答例

In [ ]:

```
# 必要なモジュールのインポート
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# データの生成
X, y = make_regression(n_samples=100, n_features=1, n_targets=1, noise=5.0, random_state=42
)
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42)

# 以下にコードを記述してください。
# モデルの構築
model = LinearRegression()
# モデルの学習
model.fit(train_X, train_y)
# test_X, test_yに対する決定係数を出力してください
print(model.score(test_X, test_y))
```

1.2.4 線形重回帰

線形重回帰 とは、予測したいデータが1つ(ex2. レストランの総合評価の点数)に対し、**予測に用いるデータが複数個(ex2. 食べ物のおいしさの点数と接客の良さの点数)** となる回帰分析です。 予測に用いられるデータ同士の関係性が薄いときには高い予測精度が得られます。

ここでも最小二乗法をもちいて予測するデータと予測に用いるデータの関係性を推定します。
重回帰の場合は予測に用いるデータを$x_0, x_1, x_2 \ldots$として

$$y = \tilde{} \ {}_0 x_0 + \tilde{} \ {}_1 x_1 + \tilde{} \ {}_2 x_2 + \cdots + \_$$

となるような $\tilde{}_0, \tilde{}_1, \tilde{}_2 \ldots, \_$ を推定することになります。

前のセッションで扱った数式に比べると、どうでしょう？xの種類が増えてそれに応じて係数（先ほどのセッション「線形単回帰」でいうa）をたくさん設定しなくてはならなくなった、というイメージがありますね。

線形重回帰もscikit-learnのlinear_modelモジュール内にあるLinearRegressionというモデルを使って回帰分析を行うことが可能です。 自動的に、すでにあるデータに対して一番フィットするように $\tilde{}_0, \tilde{}_1, \tilde{}_2 \ldots, \_$ が決定され、予測が行われます。

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# ここでn_features=10とすることでxを生成します
# 実際に使用するxの数はn_informative=3といった様に指定します
X, y = make_regression(n_samples=100, n_features=10, n_informative=3, n_targets=1, no
ise=5.0, random_state=42)
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42)

model = LinearRegression()
model.fit(train_X, train_y)
model.score(test_X, test_y)
```

問題

- 線形重回帰を実行してください。
- test_Xに対する推測結果を出力してください。
- 出力はprint関数を用いてください。

In [ ]:

```
# 必要なモジュールのインポート
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# データの生成
X, y = make_regression(n_samples=100, n_features=10, n_informative=3, n_targets=1, noise=5.0,
 random_state=42)
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42)

# 以下にコードを記述してください




# test_Xに対する推測結果を出力してください
```

ヒント

- 予測する値を算出するには model.predict(X) を用います。
- 機械学習の流れは線形単回帰とおなじです。

解答例

In [ ]:

```
# 必要なモジュールのインポート
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# データの生成
X, y = make_regression(n_samples=100, n_features=10, n_informative=3, n_targets=1, noise=5.0,
 random_state=42)
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42)

# 以下にコードを記述してください
# モデルの構築
model = LinearRegression()
# モデルの学習
model.fit(train_X, train_y)
# test_Xに対する推測結果を出力してください
print(model.predict(test_X))
```

1.3 添削問題

線形単回帰と線形重回帰を同じデータセットに対して行います。

問題

- 6種類のデータを内包する飛行機の風切り音を回帰分析するデータが渡されます。
- 予測したいデータをScaled sound pressure levelとし、Scaled sound pressure levelに対する決定係数を線形重回帰によって算出してください。
- また、各データのScaled sound pressure levelに対する決定係数を線形単回帰を用いて算出してください。

In [ ]:

```
# 必要なモジュールのインポート
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# データの読み込み
feature_names = ["Frequency", "Angle of attack", "Chord length", "Free-stream velocity",
                 "Suction side displacement thickness", "Scaled sound pressure level"]
noise_data = pd.read_csv(
    "https://archive.ics.uci.edu/ml/machine-learning-databases/00291/airfoil_self_noise.dat", sep="
\t")
noise_data.columns = feature_names

# 線形単回帰を各データについて行ってください。


# 線形重回帰用の教師データ、テストデータを準備
train_X, test_X, train_y, test_y = train_test_split(
    noise_data.drop("Scaled sound pressure level", axis=1), noise_data["Scaled sound pressure level"
], random_state=42)

# 線形重回帰を行ってください。
```

ヒント

- 各データの取り出し方は以下のようにします。

```
train_X, test_X, train_y, test_y = train_test_split(
  noise_data["Frequency"], noise_data["Scaled sound pressure level"]
# Xのshapeを修正
train_X = train_X.values.reshape((-1, 1))
test_X = test_X.values.reshape((-1, 1))
```

- データセットの詳細はこちら (https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise)をご覧ください。

解答例

In [ ]:

```
# 必要なモジュールのインポート
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# データの読み込み
feature_names = ["Frequency", "Angle of attack", "Chord length", "Free-stream velocity",
                 "Suction side displacement thickness", "Scaled sound pressure level"]
noise_data = pd.read_csv(
    "https://archive.ics.uci.edu/ml/machine-learning-databases/00291/airfoil_self_noise.dat", sep="
\t")
noise_data.columns = feature_names

# 線形単回帰を各データについて行ってください。
# for文を使用すると短いコードで処理ができます。
for name in feature_names:
    train_X, test_X, train_y, test_y = train_test_split(
        noise_data[name], noise_data["Scaled sound pressure level"], random_state=42)
    # Xのデータの形を修正
    train_X = train_X.values.reshape((-1, 1))
    test_X = test_X.values.reshape((-1, 1))
    model = LinearRegression()
    model.fit(train_X, train_y)
    print("線形単回帰に用いた変数:{}".format(name))
    print("決定係数:{}".format(model.score(test_X, test_y)))
    print()

# 線形重回帰用の教師データ、テストデータを準備
train_X, test_X, train_y, test_y = train_test_split(
    noise_data.drop("Scaled sound pressure level", axis=1), noise_data["Scaled sound pressure level"
], random_state=42)

# 線形重回帰を行ってください。
# モデルの構築
model = LinearRegression()
# モデルの学習
model.fit(train_X, train_y)
# 決定係数の出力
print("線形重回帰")
print("決定係数:{}".format(model.score(test_X, test_y)))
```

Scaled sound pressure levelは今回の予測したい値なので決定係数は必ず1になります。
それ以外の変数も単独では0.1以下ととても低い値ですが、線形重回帰によって0.5まで上昇していることから全ての変数が複雑に絡みあって
データの予測がなされていることがわかります。