クレジット:
UTokyo Online Educationデータマイニング入門 2018 森　純一郎

東京大学
THE UNIVERSITY OF TOKYO

CC BY NC ND

データ分析の実践

In [ ]:
```
import pandas as pd
import numpy as  np
import matplotlib.pyplot  as plt
%matplotlib inline
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import learning_curve
from sklearn.model_selection import validation_curve
from sklearn.metrics import mean_squared_error
```

データセットの準備

以下では、UCI Machine Learning Repositoryに公開されているワインの品質データセットを用いて、ワインの理化学検査結果に基づく特徴量からワインの品質を予測することを考えます。

データセットには赤ワインのデータセットと白ワインのデータセットが含まれますが、以下では赤ワインのデータセットを使用します。

Wine Quality Data Set (https://archive.ics.uci.edu/ml/datasets/wine+quality)

各ワインのデータは以下の11種類の特徴量からなります。

- 1 - fixed acidity
- 2 - volatile acidity
- 3 - citric acid
- 4 - residual sugar
- 5 - chlorides
- 6 - free sulfur dioxide
- 7 - total sulfur dioxide
- 8 - density
- 9 - pH
- 10 - sulphates
- 11 - alcohol

また、各ワインには0から10の品質スコアが付与さています。11種類の特徴量を元にワインの品質スコアを回帰することでワインの品質を予測することを考えます。

データの観察

まず、データセットを読み込みます。ここではデータセットのcsvファイルを pandas のデータフレームとして読み込みます。

In [ ]:
```
wine = pd.read_csv("winequality-red.csv", sep=";")
```

読み込んだデータセットのデータフレームの情報を確認してみます。1599のデータを行、11の特徴量と1つの品質スコアを列とするデータフレームとなっています。欠損値は含まれていないようです。

In [ ]:
```
wine.info()
```

データセットの表示してみます。各特徴量と品質スコアは数値で表されていることがわかります。

In [ ]:
```
wine.head()
```

各特徴量と品質スコアの記述統計を観察してみます。

In [ ]:
```
wine.describe()
```

pandas のデータフレームの可視化関数を使って各特徴量ごと値の分布を可視化してみます。

それぞれの特徴量はどのような分布になっているでしょうか。また、外れ値や特徴量の変換処理は必要でしょうか（以下では外れ値の処理を行わずに進めることとします）。

In [ ]:
```
wine.hist(bins=50, figsize=(20,20));
```

品質スコアごとのデータ数を数えると多くのデータの品質は5または6となっています。

In [ ]:
```
wine['quality'].value_counts()
```

pandas のデータフレームの corr メソッドを用いて、特徴量間および各特徴量と品質スコアの相関を計算してみます。 お互いに相関がある特徴量、品質スコアと相関がある特徴量はあるでしょうか。

In [ ]:
```
wine.corr(method='pearson')
```

In [ ]:
```
plt.figure(figsize=(7,5))
sns.heatmap(wine.corr(method='pearson'), cmap='Blues');
```

品質スコア'quiality'と特徴量'alcohol'には非常に緩やかな正の相関があることがわかります。

In [ ]:
```
corr=wine.corr(method='pearson')
corr["quality"].sort_values(ascending=False)
```

In [ ]:
```
wine.plot(kind="scatter", x="alcohol", y="quality",
          alpha=0.05, figsize=(7,5));
```

モデルの学習

線形回帰

特徴量'alcohol'から品質スコアを予測するモデルを学習することを考えます。以下では、データセットの70%をモデル学習の訓練データ、30%をモデル評価のテストデータに分割し、特徴量を標準化した上で線形回帰によるモデルの学習を行っています。学習したモデルをテストデータに適用して予測を行い、平均二乗誤差によりモデルの精度を評価しています。

In [ ]:
```
X=wine[ ['alcohol']].values
y=wine[['quality']].values

# 訓練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=1,
                                                    stratify=y,
                                                    shuffle=True)

# 標準化
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)

# 線形回帰
lr = LinearRegression()
lr=lr.fit(X_train, y_train)

# 予測
mse=mean_squared_error(y_test,lr.predict(X_test))
print(mse)
```

次に、すべての特徴量から品質スコアを予測するモデルを学習することを考えます。手順は上記と同じです。特徴量をすべて用いることでテストデータの誤差は小さくなっています。

In [ ]:
```
X=wine[ ['fixed acidity','volatile acidity','citric acid',
         'residual sugar','chlorides','free sulfur dioxide',
         'total sulfur dioxide','density', 'pH', 'sulphates',
         'alcohol']].values
# X=wine.drop("quality", axis=1).values
y=wine[['quality']].values

# 訓練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=1,
                                                    stratify=y,
                                                    shuffle=True)


# 標準化
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)

# 線形回帰
lr = LinearRegression()
lr=lr.fit(X_train, y_train)

# 予測
mse=mean_squared_error(y_test,lr.predict(X_test))
print(mse)
```

LinerRegression()オブジェクトのcoef_属性で学習された特徴量の重み（係数）を調べることができます。どの特徴量が予測に有効であると言えるでしょうか。

In [ ]:
```
pd.DataFrame(lr.coef_, index=['weight'],
             columns=wine.drop("quality", axis=1).columns.values)
```

学習曲線

学習に用いるデータを徐々に増やした時の訓練データの誤差とテストデータの誤差を以下のように学習曲線としてプロットしてみます。データを増やすとテストデータの誤差は徐々に減少していきますが、訓練データとの誤差にやや差があることがわかります。

以下のコードでは make_pipeline で標準化の処理と線形回帰モデルをまとめています。

learning_curve は train_sizes 引数で指定された分のデータを利用して交差検証（ cv 引数でfold数を指定）によりモデルの評価を行い、訓練データと検証データの予測精度（ scoring 引数で指標を指定）をそれぞれ返します。

In [ ]:
```
X=wine.drop("quality", axis=1).values
y=wine[['quality']].values

# 標準化と線形回帰モデルのパイプライン
pipe=make_pipeline(StandardScaler(), LinearRegression())

# 学習曲線
training_sizes, train_scores, valid_scores = \
                learning_curve(pipe, X, y,cv=10,
                                scoring="neg_mean_squared_error",
                                train_sizes=np.linspace(0.1, 1.0, 10))
plt.figure(figsize=(7,5))
plt.xlabel('# of instances')
plt.ylabel('MSE')
plt.plot(training_sizes, -train_scores.mean(axis=1),
         label="training")
plt.plot(training_sizes, -valid_scores.mean(axis=1),
         label="validation")
plt.legend();
```

モデル選択と評価

多項式

モデルを複雑にした時のモデル選択と評価について考えます。以下では、多項式により入力の特徴量を増やしモデルを複雑にしています。多項式の次数を増やすと訓練データの誤差は減少していきますが検証データの誤差は3次の多項式で急激に増加し、過学習がおこっていることがわかります。

validation_curve は param_name 引数で指定されたハイパーパラメータ（ここでは多項式の次数）について param_range 引数で指定した値ごとに交差検証（ cv 引数でfold数を指定）によりモデルの評価を行い、各交差検証の訓練データと検証データの予測精度（ scoring 引数で指標を指定）を返します。

In [ ]:
```
X=wine.drop("quality", axis=1).values
y=wine[['quality']].values

# 多項式特徴量と標準化と線形回帰モデルのパイプライン
pipe=make_pipeline(PolynomialFeatures(), StandardScaler(),
                   LinearRegression())

# 次数
degree = [1, 2, 3]

# バリデーション曲線
train_scores, valid_scores = \
            validation_curve(pipe, X, y,
                             param_name='polynomialfeatures__degree',
                             param_range=degree,
                             scoring="neg_mean_squared_error", cv=10)
print(-train_scores.mean(axis=1))
print(-valid_scores.mean(axis=1))

plt.figure(figsize=(7,5))
plt.xlabel('degree')
plt.ylabel('MSE')
plt.plot(degree, -train_scores.mean(axis=1),
         color='red', label='training')
plt.plot(degree, -valid_scores.mean(axis=1),
         color='blue', label='validation')
plt.legend();
```

正則化

多項式の次数を3次にした時、検証データの誤差が急激に増加し過学習がおこることを確認しました。ここで、正則化を用いて過学習を抑えることを考えます。以下では、L_2 ノルムを正則化に用いた線形回帰モデルである `Ridge` （リッジ回帰）を用いています。正則化項の係数を大きくすると、検証データの誤差が減少し、過学習が抑えられていることがわかります。一方、正則化項の係数を大きくすると、訓練データの誤差は徐々に増加していることがわかります。

In [ ]:
```
X=wine.drop("quality", axis=1).values
y=wine[['quality']].values

# 標準化と多項式特徴量（3次）とリッジ回帰モデルのパイプライン
pipe=make_pipeline(PolynomialFeatures(degree=3),
                   StandardScaler(), Ridge())

# 正則化項の係数
alpha = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

# バリデーション曲線
train_scores, valid_scores = \
            validation_curve(pipe, X, y,
                             'ridge__alpha', alpha,
                             cv=10, scoring="neg_mean_squared_error")
print(-train_scores.mean(axis=1))
print(-valid_scores.mean(axis=1))

plt.figure(figsize=(7,5))
plt.xlabel('log(alpha)')
plt.ylabel('MSE')
plt.plot(np.log10(alpha), -train_scores.mean(axis=1),
         color='red', label='training')
plt.plot(np.log10(alpha), -valid_scores.mean(axis=1),
         color='blue', label='validation')
plt.legend();
```

以下は、多項式の次数を2次にした時の正則化項の係数と誤差です。正則化項の係数を大きくすると検証データの誤差が減少しますが、大きくしすぎると、訓練データの誤差も検証データの誤差も増加するアンダーフィティング（High Bias）の状態になります。

In [ ]:
```
X=wine.drop("quality", axis=1).values
y=wine[['quality']].values

# 標準化と多項式特徴量（2次）とリッジ回帰モデルのパイプライン
pipe=make_pipeline(PolynomialFeatures(degree=2),
                   StandardScaler(), Ridge())

# 正則化項の係数
alpha = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

# バリデーション曲線
train_scores, valid_scores = \
            validation_curve(pipe, X, y,
                             'ridge__alpha', alpha,
                             cv=10, scoring="neg_mean_squared_error")
print(-train_scores.mean(axis=1))
print(-valid_scores.mean(axis=1))

plt.figure(figsize=(7,5))
plt.xlabel('log(alpha)')
plt.ylabel('MSE')
plt.plot(np.log10(alpha), -train_scores.mean(axis=1),
         color='red', label='training')
plt.plot(np.log10(alpha), -valid_scores.mean(axis=1),
         color='blue', label='validation')
plt.legend();
```

グリッドサーチ

モデルのハイパーパラメータである多項式の次数と正則化項の係数の組み合わせについて交差検証で評価を行い、モデルを選択することを考えます。グリッドサーチによるハイパーパラメータ探索の結果、学習に用いたデータセットについては、2次の多項式で正則化項の係数を100としたモデルが交差検証では最も高い精度となりました。このハイパーパラメータでモデルの学習を行い、実際の運用時の性能評価は別途用意したテストデータで評価を行います。

GridSearchCV は、 param_grid 引数で指定されたハイパーパラメータの組み合わせごとに交差検証（ cv 引数でfold数を指定）によりモデルの評価（ scoring 引数で評価指標を指定）を行います。以下ではグリッドサーチによるハイパーパラメータの組み合わせごとのモデルの予測精度をヒートマップとして可視化しています。

In [ ]:
```
X=wine.drop("quality", axis=1).values
y=wine[['quality']].values

# グリッドサーチでのパラメータのラベル
degree="param_polynomialfeatures__degree"
alpha='param_ridge__alpha'

# 標準化と多項式特徴量とリッジ回帰モデルのパイプライン
pipe = make_pipeline(PolynomialFeatures(), StandardScaler(), Ridge())

# ハイパーパラメータ（多項式の次数と正則化項の係数）
param_grid = {'polynomialfeatures__degree': [1, 2, 3],
              'ridge__alpha': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

# グリッドサーチ
grid = GridSearchCV(pipe, param_grid=param_grid,
                    scoring="neg_mean_squared_error",
                    cv=10, return_train_score=True)
grid.fit(X, y)
print(grid.best_params_)
print(-grid.best_score_)

# 可視化
results = pd.DataFrame(grid.cv_results_)
data = results[[degree, alpha, 'mean_test_score']].pivot( \
                                          index=alpha,
                                          columns=degree,
                                          values='mean_test_score')
plt.figure(figsize=(7,5))
sns.heatmap(data, annot=True, fmt='.4f',  cmap="Blues");
```

ハイパーパラメータの探索は以下のように cross_val_score を用いても行うことができます。

In [ ]:
```
X=wine.drop("quality", axis=1).values
y=wine[['quality']].values

# ハイパーパラメータ（多項式の次数と正則化項の係数）
degree=[1,2,3]
alpha=[0.01, 0.1, 1, 10, 100]

for n in degree:
    for r in alpha:
        pipe = make_pipeline(PolynomialFeatures(degree=n),
                             StandardScaler(), Ridge(alpha=r))
        scores = cross_val_score(pipe, X, y,
                                 scoring='neg_mean_squared_error',
                                 cv=10)
        print(n, r, -scores.mean())
```