#algorytmy uczenia maszynowego

#ładowanie niezbędnych pakietów

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sklearn

#przykładowy zbiór danych - pomiary fizykochemiczne wlasności portugalskich win typu Vinho Verde (białe i czerwone)

wine = pd.read\_csv("winequality-all.csv", comment="#")

wine.head()

wine.info()

#kolor wina jest typu object, więc musimy zmienić tą zmienną na zmienną kategoryczną

wine.color = wine.color.astype("category")

wine.info()

#przydatne informacje o zbiorze, które można zapisać w postaci zmiennych

wine.shape

print(wine.columns.str.cat(sep=", "))

wine.describe()

wine.iloc[:, 0:11].describe().round(1).T.iloc[:, 1:]

#-----------------

#cel - sprawdzimy czy alkohol jest funkcją pozostałych 10 zmiennych i jaka jest ta zależność

#dzięki temu będziemy w stanie wyjaśnić pochodną jakiego zbioru czynników jest dana zawartość alkoholu

#a także przewidzieć zawartość alkoholu w nowo wyprodukowanej partii wina

#sprawdzamy ile win czerwonych i ile białych jest w naszym zbiorze

wine.color.value\_counts()

#zajmiemy się winami białymi, ponieważ jest ich więcej i są słabsze :)

white\_wine = wine[wine.color == "white"]

white\_wine = white\_wine.iloc[:, 0:11]

white\_wine.head()

#tworzymy macierze zmiennych objaśniających (predyktorów) i wektor kolumnowy zmiennej objaśnianej

y = white\_wine.iloc[:, -1]

y.head()

X = white\_wine.iloc[:, :-1]

X.head()

#obliczmy wspóczynnik korelacji liniowej Pearsona

corr\_P = white\_wine.corr("pearson")

corr\_P.shape

corr\_P

#tworzymy macierz trójkątną i wyświetlamy wspóczynnik korelacji większy od 0.5

corr\_P\_tri = corr\_P.where(np.triu(np.ones(corr\_P.shape, dtype=np.bool), k=1)).stack().sort\_values()

corr\_P\_tri

corr\_P\_tri[abs(corr\_P\_tri)>0.5]

#wizualizacja skupień przy pomocy seaborn pairplot

sns.pairplot(white\_wine)

plt.show()

#algorytmy uczenia maszynowego

#tworzymy model regresji liniowej

import sklearn.linear\_model

mnk = sklearn.linear\_model.LinearRegression()

mnk.fit(X,y)

mnk.intercept\_

mnk.coef\_

white\_wine.describe()

x\_nowy = X.mean().values.reshape(1,-1)

x\_nowy

#dodajmy też małą wartość do x\_nowy, np. 0.001

mnk.predict(x\_nowy)

#ocena jakości modelu

#porównanie wartości dopasowanych, obliczonych za pomocą modelu z wartościami oryginalnymi

y\_pred = mnk.predict(X)

y\_pred[0:8]

y[0:8]

#współczynnik determinacji R2

mnk.score(X,y)

sklearn.metrics.r2\_score(y, y\_pred)

#lub inne miary błędów dopasowania

#MSE

sklearn.metrics.mean\_squared\_error(y, y\_pred)

#MAE

sklearn.metrics.mean\_absolute\_error(y, y\_pred)

#MedAE

sklearn.metrics.median\_absolute\_error(y, y\_pred)

#zależy nam na dobrych zdolnościach predykcyjnych modelu

#ale uważamy też żeby nie przeuczyć modelu,

#zatem dzielimy zbiór na próbę uczącą (80%) i testową (20%)

X\_ucz, X\_test, y\_ucz, y\_test = sklearn.model\_selection.train\_test\_split(X, y, test\_size=0.2, random\_state=12345)

print(X\_ucz.shape)

print(X\_test.shape)

print(y\_ucz.shape)

print(y\_test.shape)

#stworzymy funkcję, która dopasowuje model regresji liniowej do danej próby

#oraz oblicza miary błędów dopasowania

def fit\_regression(X\_ucz, X\_test, y\_ucz, y\_test):

r = sklearn.linear\_model.LinearRegression()

r.fit(X\_ucz, y\_ucz)

y\_ucz\_pred = r.predict(X\_ucz)

y\_test\_pred = r.predict(X\_test)

mse = sklearn.metrics.mean\_squared\_error

mae = sklearn.metrics.mean\_absolute\_error

return {

"r\_score": r.score(X\_ucz, y\_ucz),

"MSE\_u": mse(y\_ucz, y\_ucz\_pred),

"MSE\_t": mse(y\_test, y\_test\_pred),

"MAE\_u": mae(y\_ucz, y\_ucz\_pred),

"MAE\_t": mae(y\_test, y\_test\_pred)

}

#przedstawiamy działanie powyższej funkcji oraz wyniki

params = ["Reg. liniowa"]

res = [fit\_regression(X\_ucz, X\_test, y\_ucz, y\_test)]

pd.DataFrame(res, index=params)

#model wielomianowy

#korzystamy z funkcji PolynomialFeatures ze stopniem 2,

#aby wygenerować nowe cechy, które są iloczynem cech bazowych,

#np. [x1,x2,x3] -> [x1, x2, x3, x1^2, x1x2, x1x3, x2^2, x2x3, x3^2]

import sklearn.preprocessing

wielomian2\_cechy = sklearn.preprocessing.PolynomialFeatures(degree=2, include\_bias=False)

wielomian2\_cechy.fit\_transform(np.array([[2,3,5],[1,2,3]]))

#możemy sprawdzić potęgi poszczególnych zmiennych (patrzymy na kolumny)

wielomian2\_cechy.powers\_.T

#budujemy model wielomianowy przekształcając zbiór treningowy predyktorów X\_ucz

#oraz zbiór testowy predyktorów X\_test

wielomian2 = sklearn.preprocessing.PolynomialFeatures(degree=2, include\_bias=False)

X2\_ucz = wielomian2.fit\_transform(X\_ucz)

X2\_test = wielomian2.fit\_transform(X\_test)

#teraz mamy 65 kolumn

X2\_ucz.shape

#sprawdzenie działania modelu

params.append("Reg. wielomianowa")

res.append(fit\_regression(X2\_ucz, X2\_test, y\_ucz, y\_test))

pd.DataFrame(res, index=params)

#uzyskaliśmy mniejsze błędy dopasowania i predykcji, ale znacząco wzrosła liczba parametrów modelu

#redukcja zmiennych modelu

#szukamy równowagi pomiędzy złożonością modelu a jego jakością

#wyboru zmiennych do modelu możemy dokonać korzystając z kryterium Schwarza (BIC - Bayesian Information Criterion)

#wybieramy taki model regresji, który minimalizuje

#BIC(MSE\_p, p, n) = n\*log(MSE\_p) + p\*log(n)

#MSE\_p jest liczone dla modelu zbudowanego na podstawie p<=d zmiennych

#p\*log(n) to kara za złożoność modelu

#liczba wszystkich możliwych przypadków do rozpatrzenia jest rzędu 2^d

#1. zaczynamy od modelu pustego. BIC wynosi +nieskończoność

#2. rozszerzamy model o zmienną, dla której BIC jest najmniejsza i

# jednocześnie zmniejsza aktualną wartość BIC - jeśli takiej nie ma zwracamy aktualny model

#3. powtarzamy 2. aż do wyczerpania możliwości

def forward\_selection(X, y):

n, m = X.shape

best\_idx = []

best\_free = set(range(m))

best\_fit = np.inf

res = []

for i in range(0, m):

cur\_idx = -1

cur\_fit = np.inf

for e in best\_free:

r = sklearn.linear\_model.LinearRegression()

test\_idx = best\_idx + [e]

r.fit(X[:, test\_idx], y)

test\_fit = BIC(sklearn.metrics.mean\_squared\_error(y, r.predict(X[:, test\_idx])), i+2, n)

if test\_fit < cur\_fit: cur\_idx, cur\_fit = e, test\_fit

if cur\_fit > best\_fit: break

best\_idx, best\_fit = best\_idx + [cur\_idx], cur\_fit

best\_free.discard(cur\_idx)

res.append((cur\_idx, cur\_fit))

return res

#stosujemy algorytm wyboru zmiennych do zbioru przekształconego wielomianowo

#okazuje się, że wybrano 20 zmiennych

wybrane\_df = pd.DataFrame(forward\_selection(X2\_ucz, y\_ucz), columns=["zmienna", "BIC"])

wybrane\_zmienne = wybrane\_df["zmienna"].tolist()

wybrane\_df["nazwa"] = [X.columns[w>=1].append(X.columns[w==2]).str.cat(sep="\*") for w in wielomian2.powers\_[wybrane\_zmienne]]

wybrane\_df

#dodajemy kolejny wiersz do ramki danych z oceną jakości modelu

params.append("Reg. wiel. zmienne wybrane")

res.append(fit\_regression(X2\_ucz[:, wybrane\_zmienne], X2\_test[:, wybrane\_zmienne], y\_ucz, y\_test))

wyniki = pd.DataFrame(res, index=params)

wyniki

#i rysujemy wykres

wyniki.drop(["r\_score"], axis=1).plot(style=["-", ":", "--", "-."], color="k")

plt.xticks(np.arange(len(wyniki.index.tolist())), wyniki.index.tolist())

plt.show()