МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

кафедра «Вычислительная техника».

**Лабораторная работа №2 (вариант 5)**

по дисциплине: «Системы искусственного интеллекта»

Выполнил:

студент 4 курса, гр. ИВТВМбд-41

Захарычев Никита Алексеевич.

Проверил:

кандидат технических наук

Святов Кирилл Валерьевич.

г. Ульяновск, 2017

**Задание:**

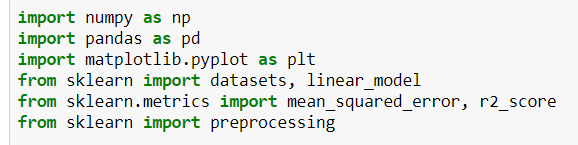
1. Разделить исходную выборку на обучающую и тестовую.
2. С использованием библиотеки scikit-learn обучить модель линейной регрессии по обучающей выборке.
3. Проверить точность модели по тестовой выборке.
4. Построить модель с использованием полиномиальной функции.
5. Построить модель с использованием регуляризации.

**Вариант 5 -** Набор данных: Forest Fires (Лесные пожары).

Набор данных представляет сжигаемую площадь лесных пожаров в северо-восточном регионе Португалии с использованием метеорологических и других данных.

**Ход работы**

Подключаем необходимые библиотеки:



**Подготовка данных**

Считываем файл с данными

data = pd.read\_csv('forestfires.csv', header=0)

Создаем вектор Y

y = data.loc[:, ["area"]]

Создаем и преобразовываем вектор X

x = data.loc[:, (data.dtypes == np.float64) | (data.dtypes == np.int64)]

Удаляем ненужные столбцы

x = data.iloc[:,:-1]

x = data.iloc[:, -9:]

Далее выводим Х:

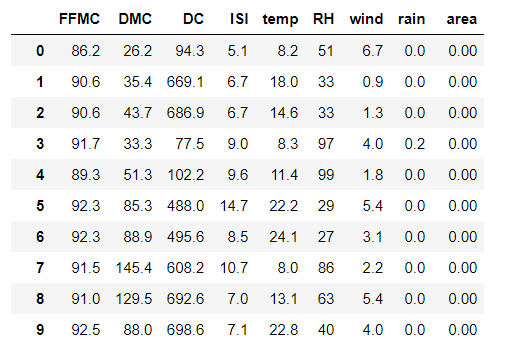


Рисунок 1. Фрагмент таблицы Х

Признаки отбирались по критерию типа. Таблица была сформирована из признаков имеющих числовой тип.

**Разделение исходной выборки на обучающую и тестовую**

train\_data = int(math.floor(all\_rows \* 0.8))

test\_data = int(math.floor(all\_rows \* 0.2))

target\_data = int(math.floor(all\_rows \* 0))

x\_train = x[:train\_data]

x\_test = x[train\_data:(train\_data + test\_data)]

y\_train = y.iloc[:train\_data]

y\_test = y.iloc[train\_data:(train\_data + test\_data)]

x\_train = preprocessing.scale(x\_train)

x\_test = preprocessing.scale(x\_test)

**Построение модели линейной регрессии**

regr = linear\_model.LinearRegression()

regr.fit(x\_train, y\_train)

y\_pred = regr.predict(x\_test)

**Среднее квадратичное отклонение**

print("Mean squared error: %.100f" % mean\_squared\_error(y\_test, y\_pred))

Mean squared error 521.876525608505630771105643361806869506835937500000000

**Коэффициент детерминации:**

print('Variance score: %.2f' % r2\_score(y\_test, y\_pred))

Variance score: 0.92

Коэффициент детерминации рассчитывается для оценки качества подбора уравнения регрессии. Для приемлемых моделей предполагается, что коэффициент детерминации должен быть хотя бы не меньше 50%. Модели с коэффициентом детерминации выше 80% можно признать достаточно хорошими. Значение коэффициента детерминации равное 1 означает функциональную зависимость между переменными, следовательно, данная модель является максимально приемлемой.

**Построение модели с использованием полиномиальной функции**

Подключим дополнительные библиотеки для генерации функции полинома и формировании соответствующих признаков:

C:\Users\Alex\Desktop\3.jpg

Подготовим данные для обучения со степенью полинома 1, так как при построении модели линейной регрессии нам удалось получить приемлемый результат.

d = [1, 2, 3, 4, 5]

varScr = []

for i in d:

polynomial\_features = PolynomialFeatures(degree=i, include\_bias=False)

pipeline = Pipeline([("polynomial\_features", polynomial\_features),

("linear\_regression", regr)])

Произведем обучение и получим значения среднего квадратичного отклонения и коэффициента детерминации:

pipeline.fit(np.array(x\_train), np.array(y\_train))

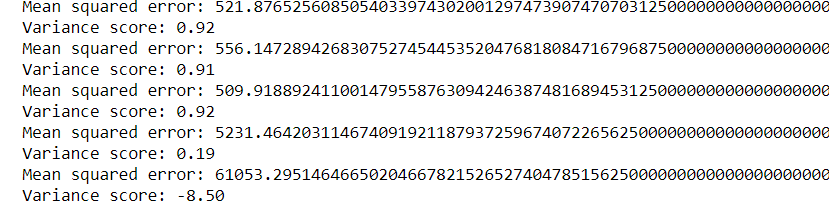
y\_pred = pipeline.predict(np.array(x\_test))

**Среднее квадратичное отклонение и коэффициент детерминации**

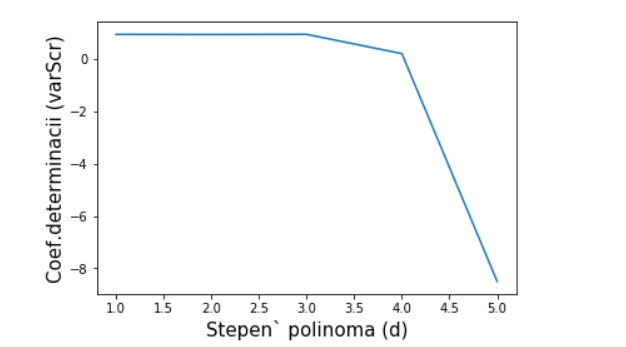
print("Mean squared error: %.100f"

% mean\_squared\_error(y\_test, y\_pred))

print('Variance score: %.2f' % r2\_score(y\_test, y\_pred))



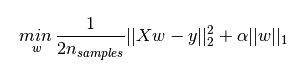
Построим график зависимости



**Построение модели с использованием регуляризации:**

В качестве модели регуляризации было принято использовать линейную модель Лассо.

Метод заключается во введении ограничения на норму вектора коэффициентов модели, что приводит к обращению в 0 некоторых коэффициентов модели. Метод приводит к повышению устойчивости модели в случае большого [числа обусловленности](http://www.machinelearning.ru/wiki/index.php?title=%D0%A1%D0%B8%D0%BD%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%BE%D0%B5_%D1%80%D0%B0%D0%B7%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) матрицы признаков X, позволяет получить интерпретируемые модели - отбираются признаки, оказывающие наибольшее влияние на вектор ответов.



Таким образом метод минимизирует средний квадрат отклонений с помощью введения коэффициента α.

В ходе выполнения работы были построены: модель линейной регрессии и полиномиальная модель со степенью полинома 1. Данные модели являются оптимальными, следовательно, можно сделать вывод, что все признаки являются значимыми, и приемлемая точность будет достигнута при α = 1.

Произведем обучение и получим значения среднего квадратичного отклонения и коэффициента детерминации:

regr = linear\_model.Lasso(alpha=1)

regr.fit(x\_train, y\_train)

y\_pred = regr.predict(x\_test)

**Среднее квадратичное отклонение**

print("Mean squared error: %.100f" % mean\_squared\_error(y\_test, y\_pred))

Mean squared error: 565.8243375216071626709890551865100860595703125000

**Коэффициент детерминации:**

print('Variance score: %.2f' % r2\_score(y\_test, y\_pred))

Variance score: 0.91

**Вывод**

Выполнив данную лабораторную работу, были улучшены навыки в программировании на языке Python. Были изучены библиотеки Python для построения модели линейной регрессии, полиномиальной модели и модели регуляризации.

**Исходный код**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn import preprocessing

# Считываем файл

data = pd.read\_csv('forestfires.csv', header=0)

# Создаем Y

y = data.loc[:, ["area"]]

# Создаем и преобразовываем X

x = data.loc[:, (data.dtypes == np.float64) | (data.dtypes == np.int64)]

# Удаляем ненужные столбцы

x = data.iloc[:,:-1]

x = data.iloc[:, -9:]

x

import math

all\_rows = 517

train\_data = int(math.floor(all\_rows \* 0.8))

test\_data = int(math.floor(all\_rows \* 0.2))

target\_data = int(math.floor(all\_rows \* 0))

x\_train = x[:train\_data]

x\_test = x[train\_data:(train\_data + test\_data)]

y\_train = y.iloc[:train\_data]

y\_test = y.iloc[train\_data:(train\_data + test\_data)]

x\_train = preprocessing.scale(x\_train)

x\_test = preprocessing.scale(x\_test)

#\_Y\_train = preprocessing.scale(\_Y\_train)

regr = linear\_model.LinearRegression()

regr.fit(x\_train, y\_train)

y\_pred = regr.predict(x\_test)

y\_pred

# Коэффициенты

print('Coefficients: \n', regr.coef\_)

# The mean squared error

print("Mean squared error: %.100f"

% mean\_squared\_error(y\_test, y\_pred))

# Explained variance score: 1 is perfect prediction

print('Variance score: %.2f' % r2\_score(y\_test, y\_pred))

y\_train

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import PolynomialFeatures

d = [1, 2, 3, 4, 5]

varScr = []

for i in d:

polynomial\_features = PolynomialFeatures(degree=i, include\_bias=False)

pipeline = Pipeline([("polynomial\_features", polynomial\_features),

("linear\_regression", regr)])

pipeline.fit(np.array(x\_train), np.array(y\_train))

y\_pred = pipeline.predict(np.array(x\_test))

# The mean squared error

print("Mean squared error: %.100f"

% mean\_squared\_error(y\_test, y\_pred))

varScr.append(r2\_score(y\_test, y\_pred))

# Explained variance score: 1 is perfect prediction

print('Variance score: %.2f' % r2\_score(y\_test, y\_pred))

i = i + 1

fig = plt.figure()

plt.plot(d,varScr)

plt.xlabel('Stepen` polinoma (d)', fontsize=15)

plt.ylabel('Coef.determinacii (varScr)', fontsize=15)

plt.show()

regr = linear\_model.Lasso(alpha=1)

regr.fit(x\_train, y\_train)

y\_pred = regr.predict(x\_test)

print("Mean squared error: %.100f"

% mean\_squared\_error(y\_test, y\_pred))

# Explained variance score: 1 is perfect prediction

print('Variance score: %.2f' % r2\_score(y\_test, y\_pred))