**Московский Государственный Технический Университет имени Н. Э. Баумана**



**Машинное обучение**

**Лабораторная работа №4**

**«Логистическая регрессия»**

ИСПОЛНИТЕЛЬ:

ФИО Гладова Анастасия

ИУ5-41 (ИУ5Ц-61Б)

ПРЕПОДАВАТЕЛЬ:

ФИО Гапанюк Ю.Е.

**Москва 2018**

# **Задание:**

Необходимо решить задачу предсказания обнаружения присутствия людей в помещении. Задача решается в рамках платформы онлайн-конкурсов по машинному обучению TrainMyData. [Ссылка на задание](https://trainmydata.com/c/occupancy_detection)

## **1. Провести предподготовку данных**

(Обязательно) Здесь можно использовать отличный туториал, предоставляемый [на сайте](https://trainmydata.com/c/occupancy_detection/discussions/page/2533274790395907). При защите нужно уметь отвечать на все вопросы, связанные с кодом.

Результатом выполнения этого пункта является блок ячеек или скрипт предобработки данных

## **2. Обучить модель из sklearn**

Следующим шагом необходимо обучить модель логистической регрессии. Для этого нужно использовать класс [LogisticRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) из sklearn.

Получить предсказания модели на валидационной части выборки. Оценить результат по метрике [Accuracy](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html).

## **3. Реализовать логистическую регрессию самостоятельно**

На этом шаге необходимо реализовать модель логистической регрессии, используя python самостоятельно. Для начала можно реализовать не векторный вариант. То есть при обучении все параметры обновлять в цикле. Этот пункт можно пропустить и переходить сразу к векторному виду, но ТОЛЬКО, если вы понимаете, как делать.

## **4. Реализовать логистическую регрессию в векторном виде**

Преобразовать код из пункта 3 в векторный формат. То есть обновление всех параметров должно происходить одновременно без циклов. Проверить, что ваш результат совпадает с результатом модели из scikit-learn.

# **Листинг:**

## **# In[1]:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cross\_validation import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_absolute\_error as mae

from sklearn.decomposition import PCA

from sklearn.metrics import accuracy\_score

import math

from sys import stdout

%matplotlib inline

# Провести предподготовку данных

data\_train = pd.read\_csv('C:/Anaconda/train.csv', delimiter=",", index\_col='id')

data\_test = pd.read\_csv('C:/Anaconda/test.csv', delimiter=",", index\_col='id')

#первые строки файла каждого датасета

data\_train.head()

## **# In[2]:**

data\_test.head()

## **# In[3]:**

#количество строк и столбцов в каждом датасете

data\_train.shape

## **# In[4]:**

data\_test.shape

## **# In[5]:**

#определим баланс классов

#количество единиц

data\_train\_count\_y1 = data\_train[data\_train['Occupancy']==1].shape[0]

#количество нулей

data\_train\_count\_y0 = data\_train[data\_train['Occupancy']==0].shape[0]

print("Количество единиц = {}, количество нулей = {}"\

.format(data\_train\_count\_y1, data\_train\_count\_y0,\

data\_train\_count\_y1 + data\_train\_count\_y0))

## **# In[6]:**

#построение корреляционной матрицы

data\_train.corr()

## **# In[7]:**

#проверка на пропущенные значения

if data\_train.columns[data\_train.isnull().values.any()].tolist() == []:

print('Нет пропущенных')

## **# In[8]:**

#Обучить модель из sklearn

#Разбиение на train и validation set(для обучения и валидации)

from sklearn.model\_selection import train\_test\_split

train, validation = train\_test\_split(data\_train, test\_size=0.2)

## **# In[9]:**

#используем метрику - точность

from sklearn.metrics import accuracy\_score

arr\_name = []

arr\_train = []

arr\_val = []

cols\_x = ['Temperature','Light','CO2','HumidityRatio']

col\_y = 'Occupancy'

def test\_classifier(classifier,classifier\_name):

classifier.fit(train[cols\_x], train[col\_y])

y\_train = classifier.predict(train[cols\_x])

y\_train\_acc = accuracy\_score(train[col\_y], y\_train)

y\_val = classifier.predict(validation[cols\_x])

y\_val\_acc = accuracy\_score(validation[col\_y], y\_val)

arr\_name.append(classifier\_name)

arr\_train.append(y\_train\_acc)

arr\_val.append(y\_val\_acc)

print('Точность для алгоритма {} на обучающей выборке={},\

на валидационной выборке ={}'\

.format(classifier\_name,\

round(y\_train\_acc, 3),\

round(y\_val\_acc, 3)))

return classifier

## **# In[10]:**

#логистическая регрессия

from sklearn.linear\_model import LogisticRegression

%time classifier = test\_classifier(LogisticRegression(),'LR' )

## **# In[11]:**

#заполним пустые количественные медианным значением

data\_train = data\_train.fillna(data\_train.median(axis=0), axis=0)

categorical\_columns = [c for c in data\_train.columns if data\_train[c].dtype.name == 'object']

numerical\_columns = [c for c in data\_train.columns if data\_train[c].dtype.name != 'object']

#заполним пустые категориальные самым частым значением по признаку

data\_describe = data\_train.describe(include=[object]) #получение сводной информации по таблице

for c in categorical\_columns:

data\_train[c] = data\_train[c].fillna(data\_describe[c]['top']) # fillna() - метод для замены отсутствующих значений на числовые

#преобразование в количественные

binary\_columns = [c for c in categorical\_columns if data\_describe[c]['unique'] == 2] #бинарные

nonbinary\_columns = [c for c in categorical\_columns if data\_describe[c]['unique'] > 2] #небинарные

data\_describe = data\_train.describe(include=[object])

for c in binary\_columns:

top = data\_describe[c]['top']

top\_items = data\_train[c] == top

data\_train.loc[top\_items, c] = 0

data\_train.loc[np.logical\_not(top\_items), c] = 1

data\_nonbinary = pd.get\_dummies(data\_train[nonbinary\_columns])

#нормализация количественных признаков

data\_numerical = data\_train[numerical\_columns]

data\_numerical = (data\_numerical - data\_numerical.mean()) / data\_numerical.std()

data\_numerical.describe()

#делаем новую таблицу с переделанными данными

data\_train = pd.concat((data\_numerical, data\_train[binary\_columns], data\_nonbinary), axis=1)

data\_train = pd.DataFrame(data\_train, dtype=int)

## **# In[12]:**

X = data\_train.drop(('Occupancy'), axis=1) # Выбрасываем столбец 'SalePrice'.

y = data\_train['Occupancy']

feature\_names = X.columns

#метод главных компонент

pca = PCA(n\_components = 5)

XPCAreduced = pca.fit\_transform(X)

#обработка данных на тренировочную и тестовую выборку

X\_train, X\_test, y\_train, y\_test = train\_test\_split(XPCAreduced, y, test\_size = 0.2, random\_state = 11)

N\_train, \_ = X\_train.shape

N\_test, \_ = X\_test.shape

#реализация библиотечной регрессии

lr = LogisticRegression()

lr.fit(X\_train, y\_train)

y\_train\_predict = lr.predict(X\_train)

y\_test\_predict = lr.predict(X\_test)

print("Модель из sklearn")

print("MAE(Средний модуль ошибки): ", mae(y\_test, y\_test\_predict))

print("Точность: ", round(accuracy\_score(y\_train, y\_train\_predict), 3))

## **# In[13]:**

#вручную

#предсказать

def predict\_outcome(feature\_matrix, weights):

weights=np.array(weights)

predictions = np.dot(feature\_matrix, weights)

return predictions

#ошибки

def errors(output,predictions):

errors=predictions-output

return errors

#производная

def feature\_derivative(errors, feature):

derivative=np.dot(2,np.dot(feature,errors))

return derivative

#градиентный спуск

def regression\_gradient\_descent(feature\_matrix, output, initial\_weights, step\_size, tolerance):

converged = False

#Начальные веса -> массив numpy

weights = np.array(initial\_weights)

while not converged:

# вычислить прогнозы

predictions=predict\_outcome(feature\_matrix,weights)

# вычислить ошибки

error=errors(output,predictions)

gradient\_sum\_squares = 0 # инициализирование градиента

# пока не сходится, обновлять каждый вес отдельно:

for i in range(len(weights)):

#вызов feature\_matrix[:, i] если столбец фич связан с весами[i]

feature=feature\_matrix[:, i]

deriv=feature\_derivative(error,feature)

#квадратная производная + градиент

gradient\_sum\_squares=gradient\_sum\_squares+(deriv\*\*2)

# обновить вес

weights[i]=weights[i] - np.dot(step\_size,deriv)

gradient\_magnitude = math.sqrt(gradient\_sum\_squares)

if gradient\_magnitude < tolerance:

converged = True

return(weights)

simple\_feature\_matrix = XPCAreduced #простая матрица харастеристик

output = y

initial\_weights = np.array([0.1, 0.001, 0.001, 0.001, 0.001])

step\_size = 0.00001

tolerance = 2.5e7 #допустимое отклонение

simple\_weights = regression\_gradient\_descent(simple\_feature\_matrix,output, initial\_weights, step\_size, tolerance)

hp = np.dot(X\_train, simple\_weights)

#сигмоида (её значение и есть предсказание)

def sigmoidfun(x):

return 1 / (1 + np.exp(-x))

hand\_y\_train\_predict = np.apply\_along\_axis(sigmoidfun, 0, hp)

hand\_y\_train\_predict = list(map(lambda x: 1 if x > 0.5 else 0, hand\_y\_train\_predict))

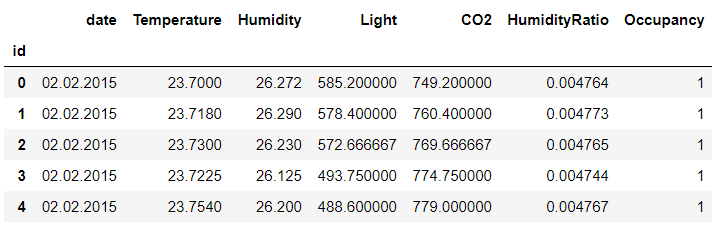
print("Вручную")

print("MAE: ", mae(y\_test, y\_test\_predict))

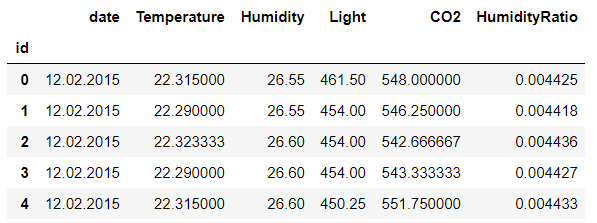
print("Точность: ", round(accuracy\_score(y\_train, hand\_y\_train\_predict, 3)))

# **Результаты выполнения:**

## **# In[1]:**



## **# In[2]:**



## **# In[3]:**

(12336, 7)

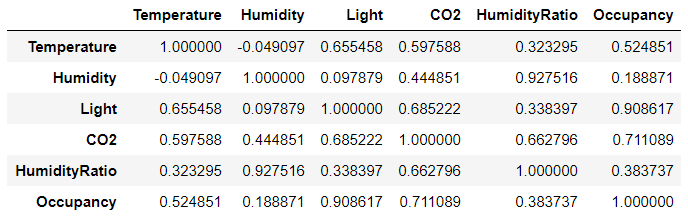
## **# In[4]:**

(8224, 6)

## **# In[5]:**

Количество единиц = 3070, количество нулей = 9266

## **# In[6]:**

****

## **# In[7]:**

Нет пропущенных

## **# In[10]:**

Точность для алгоритма LR на обучающей выборке=0.986, на валидационной выборке =0.99

Wall time: 56.8 ms

## **# In[12]:**

Модель из sklearn

MAE(Средний модуль ошибки): 0.012155591572123177

Точность: 0.986

## **# In[13]:**

Вручную

MAE: 0.012155591572123177

Точность: 1.0