Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ**

Отчет по лабораторной работе № 4
по курсу “Введение в машинное обучение”

Исполнитель:
Студент группы ИУ5-43
Желанкина А.С.
_____ 14.05.2018

Москва, 2018

Задание лабораторной работы

Необходимо решить задачу предсказания обнаружения присутствия людей в помещении. Задача решается в рамках платформы онлайн-конкурсов по машинному обучению TrainMyData.

1. Провести предподготовку данных
(Обязательно) Здесь можно использовать отличный туториал, предоставляемый на сайте. 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 из sklearn.

Получить предсказания модели на валидационной части выборки. Оценить результат по метрике Accuracy.

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.

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.

Текст программы

```
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

#открыть данные

filename = 'C:/anaconda/train_lr.csv'
```

```
data = pd.read_csv(filename)
#print(data.head(5))

#избавиться от отсутствующих

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

categorical_columns = [c for c in data.columns if data[c].dtype.name ==
'object']
numerical_columns   = [c for c in data.columns if data[c].dtype.name !=
'object']
data_describe = data.describe(include=[object])
for c in categorical_columns:
    data[c] = data[c].fillna(data_describe[c]['top'])

#преобразование в количественные

binary_columns    = [c for c in categorical_columns if
data_describe[c]['unique'] == 2]
nonbinary_columns = [c for c in categorical_columns if
data_describe[c]['unique'] > 2]

#print(binary_columns)
#print(nonbinary_columns)

#bin

data_describe = data.describe(include=[object])

for c in binary_columns:
    top = data_describe[c]['top']
    top_items = data[c] == top
    data.loc[top_items, c] = 0
    data.loc[np.logical_not(top_items), c] = 1

#nonbin

data_nonbinary = pd.get_dummies(data[nonbinary_columns])
#print(data_nonbinary.columns)

#нормализэйшн

data_numerical = data[numerical_columns]
data_numerical = (data_numerical - data_numerical.mean()) /
data_numerical.std()
data_numerical.describe()

#делаем новую таблицу с переделанными данными

data = pd.concat((data_numerical, data[binary_columns], data_nonbinary),
axis=1)
data = pd.DataFrame(data, dtype=int)
#print(data.shape)
#print(data.columns)


X = data.drop(('Occupancy'), axis=1)  # Выбрасываем столбец 'SalePrice'.
y = data['Occupancy']
feature_names = X.columns
```

```
#метод главных компонент

pca = PCA(n_components = 5)
XPCAreduced = pca.fit_transform(X)
#print(XPCAreduced)

#print(feature_names)

#обраотка данных на тренировочную и тестовую

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
#print(N_train, N_test)

#реализация библиотечного

lr = LogisticRegression()
lr.fit(X_train, y_train)

y_train_predict = lr.predict(X_train)
y_test_predict = lr.predict(X_test)

#print(y_train_predict)
#print(y_test_predict)

print("sklearn")
print("MAE: ", mae(y_test, y_test_predict))
print("Accuracy test: ", round(accuracy_score(y_train, y_train_predict), 3))

#ручками

#предсказать
def predict_outcome(feature_matrix, weights):
    weights=np.array(weights)
    predictions = np.dot(feature_matrix, weights)
    return predictions

#ошибочки
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


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
output = y
initial_weights = np.array([0.1, 0.001, 0.001, 0.001, 0.001])
step_size = 0.00001
tolerance = 2.5e7 #доустимое отклонение
simple_weights = regression_gradient_descent(simple_feature_matrix, output,
initial_weights, step_size, tolerance)
print(simple_weights)

hp = np.dot(X_train, simple_weights)
#print(type(hp))
#hp = np.dot(X_test, simple_weights)

def sigmoidfun(x):
    return 1 / (1 + np.exp(-x))

hand_y_train_predict = np.apply_along_axis(sigmoidfun, 0, hp)
print(hand_y_train_predict)

hand_y_train_predict = list(map(lambda x: 1 if x > 0.5 else 0,
hand_y_train_predict))

#hand_y_train_predict = np.apply_along_axis(lambda x: 1 if x > 0.5 else 0, 0,
hand_y_train_predict)
#print(hand_y_train_predict)

print("hands")
print("MAE: ", mae(y_test, y_test_predict))
print("Accuracy test: ", round(accuracy_score(y_train, hand_y_train_predict,
3)))
```

Результат выполнения

```
sklearn
MAE:  0.012155591572123177
Accuracy test:  0.986
[ 0.12347832  0.04736171 -0.00073854  0.01666589  0.02072685]
[0.48572335 0.65997726 0.49039701 ... 0.49355206 0.55038492 0.52423516]
hands
MAE:  0.012155591572123177
Accuracy test:  1.0
```