Московский государственный технический университет им. Н. Э. Баумана

***Факультет** «Информатика и системы управления»*

***Кафедра** «Системы обработки информации и управления»*

Отчёт по лабораторной работе №4

*по курсу «**Введение в машинное обучение**»*

Исполнила: Алиева Д.Г., ИУ5-41
Проверил: Гапанюк Ю.Е.

Москва 2018 г.

Задание ЛР:

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:

In [6]:
```
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
```

In [10]:
```
#открыть данные
data = pd.read_csv('./KaggleLab4/train.csv')
```

In [11]:
```
data
```

Out[11]:

	id	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
0	0	02.02.2015	23.700000	26.272000	585.200000	749.200000	0.004764	1
1	1	02.02.2015	23.718000	26.290000	578.400000	760.400000	0.004773	1
2	2	02.02.2015	23.730000	26.230000	572.666667	769.666667	0.004765	1
3	3	02.02.2015	23.722500	26.125000	493.750000	774.750000	0.004744	1
4	4	02.02.2015	23.754000	26.200000	488.600000	779.000000	0.004767	1
5	5	02.02.2015	23.760000	26.260000	568.666667	790.000000	0.004779	1
6	6	02.02.2015	23.730000	26.290000	536.333333	798.000000	0.004776	1
7	7	02.02.2015	23.754000	26.290000	509.000000	797.000000	0.004783	1
8	8	02.02.2015	23.754000	26.350000	476.000000	803.200000	0.004794	1
9	9	02.02.2015	23.736000	26.390000	510.000000	809.000000	0.004796	1
10	10	02.02.2015	23.745000	26.445000	481.500000	815.250000	0.004809	1
11	11	02.02.2015	23.700000	26.560000	481.800000	824.000000	0.004817	1
12	12	02.02.2015	23.700000	26.600000	475.250000	832.000000	0.004824	1
13	13	02.02.2015	23.700000	26.700000	469.000000	845.333333	0.004842	1
14	14	02.02.2015	23.700000	26.774000	464.000000	852.400000	0.004856	1
15	15	02.02.2015	23.700000	26.890000	464.000000	861.000000	0.004877	1
16	16	02.02.2015	23.700000	26.972500	455.000000	880.000000	0.004892	1
17	17	02.02.2015	23.600000	26.890000	454.000000	891.000000	0.004848	1
18	18	02.02.2015	23.640000	26.976000	458.000000	897.600000	0.004875	1
19	19	02.02.2015	23.650000	27.050000	464.000000	900.500000	0.004891	1
20	20	02.02.2015	23.640000	27.100000	473.000000	908.800000	0.004898	1

In [14]:
```
#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'])
```

In [15]:
```
#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]
```

In [17]:
```
print(binary_columns)
```

```
[]
```

In [18]:
```
print(nonbinary_columns)
```

```
['date']
```

In [20]:
```
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
```

In [21]:
```
#nonbin

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

In [22]:
```
#нормализация

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

Out[22]:

	id	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
count	1.233600e+04	1.233600e+04	1.233600e+04	1.233600e+04	1.233600e+04	1.233600e+04	1.233600e+04
mean	4.553930e-17	-2.252059e-12	-4.082321e-13	2.015519e-15	2.383677e-15	-1.612468e-14	1.090081e-14
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.731840e+00	-1.686939e+00	-1.883572e+00	-6.576697e-01	-7.517388e-01	-1.627543e+00	-5.755797e-01
25%	-8.659201e-01	-6.343865e-01	-7.161261e-01	-6.576697e-01	-6.504340e-01	-7.660362e-01	-5.755797e-01
50%	0.000000e+00	-1.958228e-01	-1.923782e-02	-6.576697e-01	-5.368099e-01	-1.261253e-01	-5.755797e-01
75%	8.659201e-01	6.813046e-01	5.913290e-01	1.275745e+00	4.235229e-01	6.239514e-01	-5.755797e-01
max	1.731840e+00	3.056858e+00	2.826740e+00	7.303243e+00	4.659279e+00	3.274235e+00	1.737238e+00

In [23]:
```
#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)
```

In [24]:
```
X = data.drop(('Occupancy'), axis=1)  # Выбрасываем столбец 'SalePrice'.
y = data['Occupancy']
feature_names = X.columns
```

In [25]:
```
#метод главных компонент

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

#print(feature_names)
```

In [26]:
```
#обработка 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)
```

In [27]:
```
#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))
```

```
sklearn
MAE:  0.0121555915721
Accuracy test:  0.986
```

In [28]:
```
#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
```

In [29]:
```
#производная
def feature_derivative(errors, feature):
    derivative=np.dot(2,np.dot(feature,errors))
    return derivative
```

In [30]:
```
def regression_gradient_descent(feature_matrix, output, initial_weights, step_size, tolerance):
    converged = False
    #Начальные веса -> массив питру
    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)
```

In [31]:
```
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))
```

```
[ 0.12347832  0.04736171 -0.00073854  0.01666589  0.02072685]
[ 0.48572335  0.65997726  0.49039701 ...,  0.49355206  0.55038492
  0.52423516]
```

In [32]:
```
#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)))
```

```
hands
MAE:  0.0121555915721
Accuracy test:  1.0
```