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 № 3 по курсу “Введение в машинное обучение”

Исполнитель:
Студент группы ИУ5-43
Желанкина А.С.
_____ 06.05.2018

Москва, 2018

Задание лабораторной работы

Необходимо решить задачу предсказания стоимости дома в зависимости от его характеристик. Задача решается в рамках платформы онлайн-конкурсов по машинному обучению Kaggle. Ссылка на задание.

Рекомендуется перед выполнением задания изучить отличный туториал, предоставляемый на сайте. В нем разбирается задача классификации, что соответствует лабораторной работе №4, но все действия связанные с подготовкой данных полностью актуальны для этой лабораторной работы.

1. Провести предподготовку данных
Перед выполнением этого пункта рекомендуется посмотреть лекцию по Pandas.

(Обязательно) Необходимо перевести категориальные фичи в числовые, отмасштабировать показатели для лучшей обучаемости модели при необходимости (можно провести эксперименты, как это будет влиять на результаты модели). Построить графики по распределнию площадей домой и распределнию цен. Для реализации этой части использовать библиотеки pandas и matplotlib и seaborn.

Необходимо оценить предоставляемые данные, на свое усмотрение предположить несколько возможных зависимостей между признаками и предсказываемыми значениями, проверить гипотезы, построив необходимые графики.

По возможности можно определить, какие признаки являются незначимыми или их доля мала, и объединить такие признаки с другими.

Создать несколько собственных фич на основе своих эвристик и оценить, влияют ли они на качество модели.

Результатом выполнения этого пункта является блок ячеек или скрипт предобработки данных.

2. Разделить данные
В этом пункте необходимо поделить данные на обучающую и валидационную выборку. Для этого можно использовать train_test_split. Делить можно в соотношениях 70-90 / 30-10 % соответственно.

3. Обучить модель из sklearn
Следующим шагом необходимо обучить модель линейной регрессии. Для этого нужно использовать класс LinearRegression из sklearn.

Получить предсказания модели на валидационной части выборки. Оценить результат по метрике Mean Absolute Error (MAE) и по метрике, используемой для оценки результатов этого конкурса на kaggle.

4. Реализовать линейную регрессию
На этом шаге необходимо реализовать модель линейной регрессии, используя python самостоятельно. Для этого изначально можно попробовать написать алгоритм для одного обучаемого параметра, а затем написать реализацию общего случая, используя сначала циклы, а затем векторные вычисления из библиотеку numpy. Если есть полное понимание, как нужно реализовать алгоритм для множества обучаемых параметров с использованием векторизации вычислений - можно сразу делать так, главное на защите уметь объяснить.

5. Эксперименты с моделью
На этом шаге вы уже имеете базовую модель, которая делает предсказания. Необходимо прогнать модель на тестовой выборке и отправить решение на kaggle. После этого можно улучшать свой результат, экспериментируя с подготовкой данных и параметрами модели. Рекомендуется смотреть т.н. kernel'ы на kaggle - раздел, где участники соревнований выкладывают код со своими идеями и реализациями. Это может быть очень полезно, как для обучения, так и для реализации новых идей.

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 LinearRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error as mae
from pandas.plotting import scatter_matrix
from sklearn.decomposition import PCA
import math
from sys import stdout
%matplotlib inline

def rmsle(y_true, y_pred):
    assert len(y_true) == len(y_pred)
    return np.square(np.log(y_pred + 1) - np.log(y_true + 1)).mean() ** 0.5

#открыть данные

filename = 'C:/anaconda/train.csv'
data = pd.read_csv(filename)

plt.figure(figsize = (8,5))
plt.scatter(data['GrLivArea'], data['SalePrice'])
#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=float)
#print(data.shape)
#print(data.columns)

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

#метод главных компонент
```

```
pca = PCA(n_components = 7)
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
= 0.3, random_state = 11)

N_train, _ = X_train.shape
N_test,  _ = X_test.shape
#print(N_train, N_test)

#реализация библиотечного

lr = LinearRegression()
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("RMSE: ", rmsle(y_test, y_test_predict))

#реализация ручками

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

def errors(output,predictions):
    errors=predictions-output
    return errors

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 преобразуются в массив numpy
    weights = np.array(initial_weights)
    while not converged:
        # вычислить прогнозы на основе feature_matrix и весов:
        predictions=predict_outcome(feature_matrix,weights)
        # вычислять ошибки как predictions - output:
        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)
            # добавить квадратную производную к величине градиента
            gradient_sum_squares=gradient_sum_squares+(deriv**2)
            # обновить вес на основе размера шага и производной:
            weights[i]=weights[i] - np.dot(step_size,deriv)

        gradient_magnitude = math.sqrt(gradient_sum_squares)
        #stdout.write("\r%d" % int(gradient_magnitude))
        stdout.flush()
        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, 0.001, 0.001])
step_size = 0.00001
tolerance = 2.5e7
simple_weights = regression_gradient_descent(simple_feature_matrix, output,
initial_weights, step_size, tolerance)
#print(simple_weights)

hand_y_train_predict = np.dot(X_train, simple_weights)
hand_y_test_predict = np.dot(X_test, simple_weights)

print("hands")
print("MAE: ", mae(y_test, hand_y_test_predict))
print("RMSE: ", rmsle(y_test, hand_y_test_predict))
```

Результат выполнения

```
sklearn
MAE:  0.26824497560865296
RMSE:  0.4956133509159565
hands
MAE:  0.42760077681552056
RMSE:  0.668958927295299
```