Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Практическая работа

по дисциплине:

Визуализация и моделирование

Авторы: Редичкина Анна Максимовна,
Походня Ксения Дмитриевна

Группа: ВИМ 1.1

Факультет: ИКТ

Преподаватель: Чернышева А.В

Санкт-Петербург

2021

Описание датасета

Датасет содержит в себе информацию об академической успеваемости школьников, их учебных достижениях. На основе этих параметров был рассчитан шанс на поступление в один из пяти университетов. В датасете 8 колонов и 500 строк
Код:

```
df = pd.read_csv('Admission_Predict_Ver1.1.csv')
df.head()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

Линейная регрессия

Многие алгоритмы машинного обучения работают лучше, когда параметры имеют относительно одинаковый масштаб. Воспользуемся методом MinMaxScaler.Для каждого значения в объекте MinMaxScaler вычитает минимальное значение в объекте и затем делит на диапазон. Диапазон - это разница между исходным максимумом и исходным минимумом.
Код:

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler= MinMaxScaler()
```

```
num_vars = ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
        'Chance of Admit ']

df[num_vars] = scaler.fit_transform(df[num_vars])
```

Датасет с нормализованными данными

```
df.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	0.94	0.928571	0.75	0.875	0.875	0.913462	1	0.920635
1	0.68	0.535714	0.75	0.750	0.875	0.663462	1	0.666667
2	0.52	0.428571	0.50	0.500	0.625	0.384615	1	0.603175
3	0.64	0.642857	0.50	0.625	0.375	0.599359	1	0.730159
4	0.48	0.392857	0.25	0.250	0.500	0.451923	0	0.492063

Зависимым параметром будут данные из колонки 'Chance of Admit', а независимыми параметрами станут оставшиеся колонки (так как именно они влияют на шансы на поступление).

```
df_y=df['Chance of Admit ']
df.drop('Chance of Admit ', axis=1,inplace=True)
```

Разделим датасет на обучающий и тестовый

```
X_train, X_test, y_train, y_test = train_test_split(df, df_y, test_size=0.3,random_state=80)
```

```
reg = LinearRegression().fit(X_train, y_train)
y_pred = reg.predict(X_test)
```

Коэффициент детерминации указывает насколько тесной является связь между факторами регрессии и зависимой переменной. В нашем случае модель объя
вариацию зависимой модели на 83%.

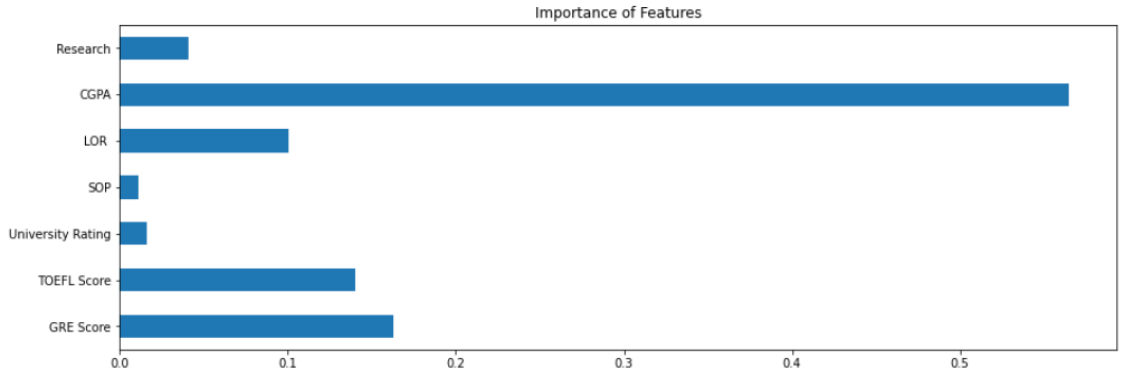
```
print('R Squared: {:.2f}'.format(reg.score(X_test, y_test)*100))
```

```
R Squared: 83.35
```

Построим график, который показывает насколько сильно каждый из параметров влияет на зависимую переменную:

```
print(reg.coef_)
importance=reg.coef_
importance=pd.Series(reg.coef_,index =['GRE Score','TOEFL Score','University Rating','SOP','LOR ','CGPA','Research'])
importance.plot(kind='barh',title='Importance of Features',figsize=(15,5))
```

```
[0.16317999 0.1403648  0.01627883 0.01105758 0.10073275 0.56493183
 0.04130361]
<AxesSubplot:title={'center':'Importance of Features'}>
```



Логистическая регрессия

Логистическая регрессия это статистический метод для анализа набора данных, в котором есть одна или несколько независимых переменных, которые определяют результат. Результат измеряется с помощью дихотомической переменной (в которой есть только два возможных результата).
Так как возможных вариантов должно быть только два, то введем колонку 'Probability of Acceptance', которая принимает значения 0 и 1. Если шансы на поступление у абитуриента выше медианного, то значение равно 1, а если меньше, то 0
Код:

```
df_y.median()
```

```
0.6031746031746031
```

```
targets = np.where(df_y>= df_y.median(), 1, 0)
targets.shape
```

```
(500,)
```

```
df['Probability of Acceptance'] = targets
df.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Probability of Acceptance
0	0.94	0.928571	0.75	0.875	0.875	0.913462	1	1
1	0.68	0.535714	0.75	0.750	0.875	0.663462	1	1
2	0.52	0.428571	0.50	0.500	0.625	0.384615	1	1
3	0.64	0.642857	0.50	0.625	0.375	0.599359	1	1
4	0.48	0.392857	0.25	0.250	0.500	0.451923	0	0

Обучим модель:

```
x_train, x_test, y_train, y_test = train_test_split(inputs, targets, test_size=0.3, random_state=20)
```

Обучим модель

```
log_reg = LogisticRegression()
log_reg.fit(x_train, y_train)
```

```
LogisticRegression()
```

Модель дает правильный результат в 87% случаев

```
log_reg.score(x_train, y_train)
```

```
0.8771428571428571
```

```
log_reg.intercept_
```

```
array([-6.50426651])
```

```
log_reg.coef_
```

```
array([[2.72488349, 2.01526692, 0.82233502, 1.68993416, 1.37214214,
        2.58811911, 0.98359133]])
```

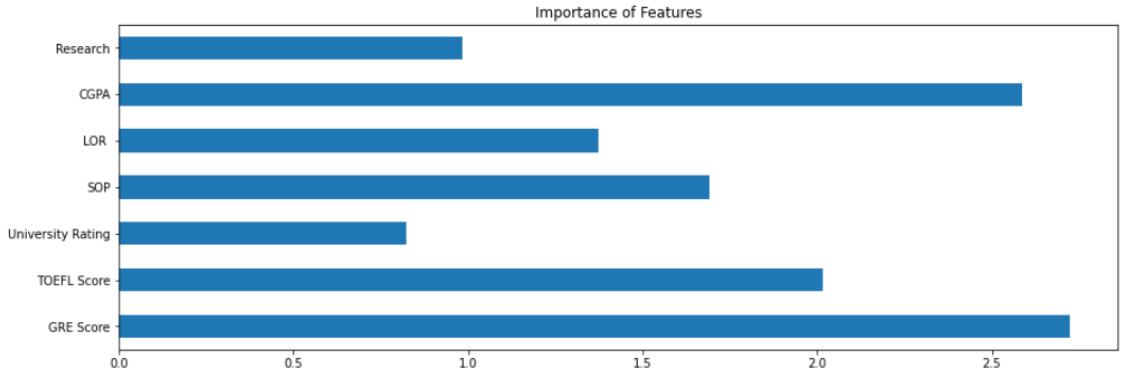
```
log_reg.score(x_test, y_test)
```

```
0.8733333333333333
```

Построим график значимости каждого из независимых параметров:

```
print(log_reg.coef_)
importance=log_reg.coef_
importance=pd.Series(log_reg.coef_[0],index =['GRE Score','TOEFL Score','University Rating','SOP','LOR ','CGPA','Research'])
importance.plot(kind='barh',title='Importance of Features',figsize=(15,5))
```

```
[[2.72488349 2.01526692 0.82233502 1.68993416 1.37214214 2.58811911
  0.98359133]]
<AxesSubplot:title={'center':'Importance of Features'}>
```



Метод ближайших соседей

Разделяем датасет на тренировочный и обучающий и выбираем различное количество соседей, чтобы сравнить результаты:

```
X_train, X_test, y_train, y_test = train_test_split(df.to_numpy()[:,1:-1], df['Probability of Acceptance'].to_numpy(), test_size=0.7,random_state=20)
```

Выбираем различное количество соседей, чтобы сравнить результаты

```
kNN_1 = KNeighborsClassifier(n_neighbors=1)
kNN_2 = KNeighborsClassifier(n_neighbors=2)
kNN_3 = KNeighborsClassifier(n_neighbors=3)
kNN_4 = KNeighborsClassifier(n_neighbors=4)
kNN_5 = KNeighborsClassifier(n_neighbors=5)
```

Результаты по каждому количеству соседей:

```
kNN_1.fit(X_train, y_train)
print(classification_report(y_test, kNN_1.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.84      0.73      0.78       175
           1       0.76      0.86      0.80       175

    accuracy                           0.79       350
   macro avg       0.80      0.79      0.79       350
weighted avg       0.80      0.79      0.79       350
```

```
kNN_2.fit(X_train, y_train)
print(classification_report(y_test, kNN_2.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.79      0.84      0.81       175
           1       0.83      0.77      0.80       175

    accuracy                           0.81       350
   macro avg       0.81      0.81      0.81       350
weighted avg       0.81      0.81      0.81       350
```

```
kNN_3.fit(X_train, y_train)
print(classification_report(y_test, kNN_3.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.89      0.74      0.81       175
           1       0.78      0.91      0.84       175

    accuracy                           0.83       350
   macro avg       0.83      0.83      0.82       350
weighted avg       0.83      0.83      0.82       350
```

```
kNN_4.fit(X_train, y_train)
print(classification_report(y_test, kNN_4.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.80      0.83       175
           1       0.81      0.86      0.84       175

    accuracy                           0.83       350
   macro avg       0.83      0.83      0.83       350
weighted avg       0.83      0.83      0.83       350
```

```
kNN_5.fit(X_train, y_train)
print(classification_report(y_test, kNN_5.predict(X_test)))
```

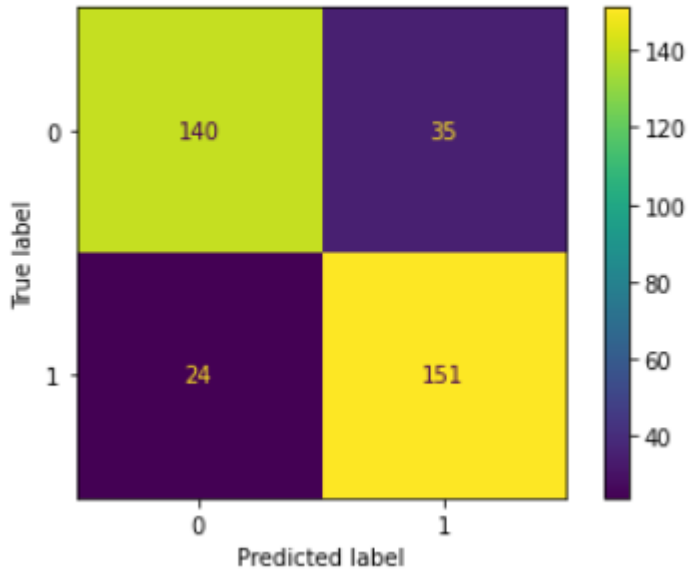
```
              precision    recall  f1-score   support

           0       0.90      0.73      0.81       175
           1       0.77      0.92      0.84       175

    accuracy                           0.83       350
   macro avg       0.84      0.83      0.82       350
weighted avg       0.84      0.83      0.82       350
```

С увеличением количества соседей увеличивается точность классификация. Для построения графика воспользуемся классификацией с kNN = 4 т.к метрики precision и recall примерно одинаково высокие, а accuracy у классификации с четырьмя и пятью соседями одинакова.

```
plot_confusion_matrix(kNN_4, X_test, y_test);
```



Наивный байесовский классификатор

Наивный байесовский алгоритм – это алгоритм классификации, основанный на теореме Байеса с допущением о независимости признаков. Другими словами, НБА предполагает, что наличие какого-либо признака в классе не связано с наличием какого-либо другого признака.
Классами здесь также являются значения колонки 'Probability of Acceptance'.

```
X_train, X_test, y_train, y_test = train_test_split(df.to_numpy()[:,1:-1], df['Probability of Acceptance'].to_numpy(), test_size=0.4,random_state=50)
```

```
clf = MultinomialNB()
clf.fit(X_train, y_train)
```

```
MultinomialNB()
```

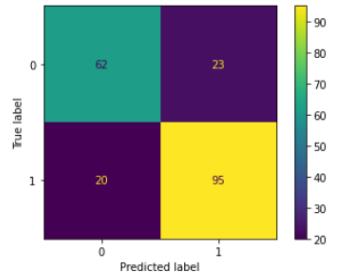
```
pred = clf.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.73      0.74        85
           1       0.81      0.83      0.82       115

    accuracy                           0.79       200
   macro avg       0.78      0.78      0.78       200
weighted avg       0.78      0.79      0.78       200
```

```
plot_confusion_matrix(clf, X_test, y_test);
```



Модель дает верные результаты с точностью в 79%