Research on the Prediction Model of Admission Rate Based on Machine Learning Model

Yanan Zhou
Chao Li
Xueyan Wang
2020.12.10

Catalog

Research on the Prediction Model of Admission Rate Based on Machine Learning Model

Abstract: Today, the tide of studying abroad continues to sweep, the TOEFL test, the GRE test, and the achievement point are the standardized indicators that the "studying abroad party" often hangs in the mouth. Applying for colleges and universities can not travel far to understand the students' ability, can only measure a student's academic level and ability through standardized indicators. For this reason, undergraduate students try their best to optimize their indicators only to win the "goddess school" favor. In this paper, K nearest neighbor model, stochastic forest model and linear regression model which can be realized in machine learning are used to predict the admission rate of American graduate students according to the standardized indexes such as TOEFL score, GRE score, CGPA, SOP and LOR.

Keywords: Study abroad; Standardized indicators; Machine learning; Admission rate forecast

0 Introduction

Since the 1980 s, the demand for studying abroad has been increasing. According to the data of the Ministry of Education of China, the total number of Chinese students studying abroad in 1978-2016 years reached 4.5866 million, and the number of Chinese students studying abroad in 2016 was 544500, a year-on-year increase of 3.97%. In 2017, the number of Chinese students studying abroad for the first time broke through the 600000 mark, reaching 608400, a year-on-year increase of 11.74%, and maintaining the status of the largest source country of foreign students in the world.

The United States is still the first choice for the destination of Chinese students. According to the Open Portal Report of the United States in 2018, the number of Chinese students from the United States in the academic year from 2017 to the 18th increased by 3.6%, accounting for 33.2% of the total number of international students in the United States.

As everyone knows, the application for studying abroad, especially higher education, for example, the application for graduate students is a "three-fee" process - expensive, time-consuming, painstaking. Some undergraduate students from the first grade very hard study professional course, improve the GPA; Take the TOEFL exam, if the score is not ideal will continue to participate many times; Finally to take the GRE exam, invested a lot of time and money. Still, they are confused about whether they can apply for dream school. At this point, a model that predicts applicant admissions based on data showing applicant capabilities such as GPAs, TOEFL, and GREs is important for students who are confused. In this study, we will comprehensively analyze the data, through the establishment and optimization of six models based on the machine learning course, in order to find an optimal predictive admission rate model.

1 Exploratory data analysis

1.1 Source of data set

Kaggle is a data modeling and data analysis competition platform. Enterprises and researchers can publish data on it for statisticians and data mining experts to compete on it to produce the best models. Many of the strategies proposed by the researchers can be used to solve almost all the problems of predictive modeling. We downloaded a dataset of undergraduate applications for admission to the American Academy on the kaggle platform, and the following are the details of the dataset:

NUMBER OF SAMPLES: 400
GRE: GRE exam score (full score 340)
TOEFL: TOEFL exam score (full score 120)

University Rating: Relative ranking of undergraduate universities (scoring system, full score of five points)
SOP: Application Document (Statement of Purpose) (full score 5pts)
LOR: Recommendation Letter (Letter of Recommendation) (full score 5pts)
CGPA: Core Core GPA (10%)
Research: Whether there is scientific research experience, 0 means none, and 1 means there is
Chance of Admit: The student's chances of admission to the institution (0-1)

The data set has a sample of 400 students and uses seven important characteristics to predict the probability that applicants will be admitted to the school. These seven characteristics are :GRE test score, full score 340; TOEFL Exam score, full score 120; Relative ranking of undergraduate universities, scoring system, higher scores ranked higher; Application documents and recommendation letter are also scoring system, full score 5 points, the higher the score, the better the quality of documents. Core courses GPA, full score 10; Scientific research experience, 0 indicates no scientific research experience, 1 indicates scientific research experience. Predicted Tag Chance of Admit: The student's chance of being admitted to the institution is between 0~1.

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65
5	330	115	5	4.5	3.0	9.34	1	0.90
6	321	109	3	3.0	4.0	8.20	1	0.75
7	308	101	2	3.0	4.0	7.90	0	0.68
8	302	102	1	2.0	1.5	8.00	0	0.50
9	323	108	3	3.5	3.0	8.60	0	0.45

Top 10 data displays

Based on the comprehensive conditions of these applicants and the understanding of graduate applications in the United States on weekdays, we deduce that this school is one of the more representative top50 schools in the United States.

1.2 Data set exploration

First we need to have a general understanding of the data, preliminary data analysis, specific code and explanation as follows:

```
# Import data
    import os
    os.chdir('C:/temp')
# Database required for loading data analysis
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
# Load Dataset
    data = pd.read_csv('uni.csv')
# Display top ten data
    data.head(10)
# Size of dataset
    data.shape
# Check datasets for missing values
```

```
    print(data.isnull().sum())
# View the data type for each data characteristic
    Print(data.dtypes)
# View descriptive statistics for datasets
    data.describe(include='all')
```

```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

After we looked at the missing values, we found that the dataset was not missing.

```
GRE Score              int64
TOEFL Score            int64
University Rating      int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
Chance of Admit      float64
dtype: object
```

The label of the prediction model is the floating point number

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chan A
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.00
mean	316.807500	107.410000	3.087500	3.400000	3.452500	8.598925	0.547500	0.72
std	11.473646	6.069514	1.143728	1.006869	0.898478	0.596317	0.498362	0.14
min	290.000000	92.000000	1.000000	1.000000	1.000000	6.800000	0.000000	0.34
25%	308.000000	103.000000	2.000000	2.500000	3.000000	8.170000	0.000000	0.64
50%	317.000000	107.000000	3.000000	3.500000	3.500000	8.610000	1.000000	0.73
75%	325.000000	112.000000	4.000000	4.000000	4.000000	9.062500	1.000000	0.83
max	340.000000	120.000000	5.000000	5.000000	5.000000	9.920000	1.000000	0.97

Descriptive statistics show us the maximum, minimum, and average values for each parameter. For example, of the 400 applicants, the lowest TOEFL score was 92, the highest was 120, and the average TOEFL score was 107.41.

1.3 Dataset Visualization

Data visualization refers to the visual and interactive display of relatively obscure data so as to visualize and intuitively express the information and laws contained in the data. Charts can more intuitively display data information than numbers, allowing readers to quickly understand the relationship between tags and features. The data set visualization code we used is as follows:

```
# Display the correlation matrix of parameters
    corr = data[['Chance of Admit','GRE Score','TOEFL Score', 'CGPA','SOP','Research']].corr()
```

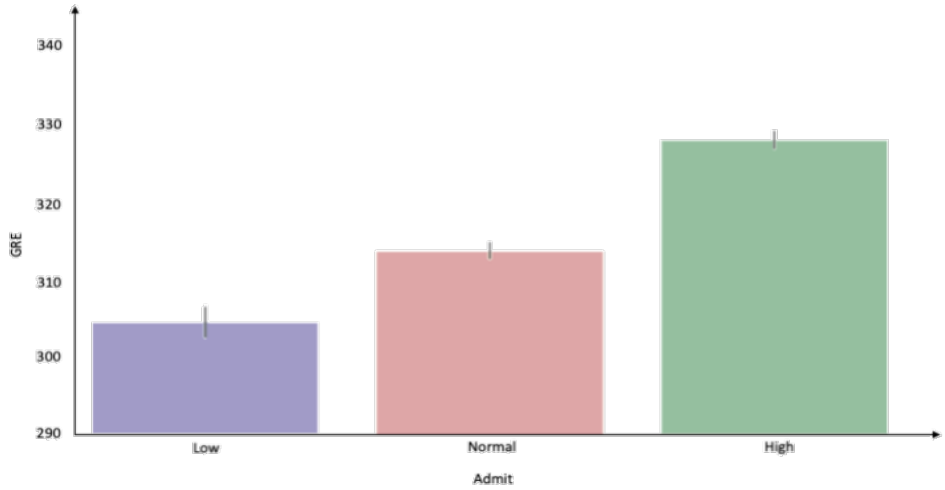
```
corr
# Change numeric type, grouping of feature 'Adimit' for later drawing
def change_to_categorical(x):
if x<=0.60:
    return 'low'
    elif x>=0.80:
    return 'high'
 else:
    return 'normal'
 data['Admit'] = data.apply(lambda row: change_to_categorical(row['Chance of Admit']), axis=1)
 data = data.drop(['Chance of Admit'],axis = 1)
 # Shows the relationship between GRE scores and admission probability
 sns.set(rc={'figure.figsize':(20,10)})
 sns.barplot(x = 'Admit',y = 'GRE Score',    order = ['low','normal','high'],data = data)
 plt.ylim(290, 340)
 # Shows the relationship between relative ranking of universities and admissions
 sns.set(rc = {'figure.figsize':(20,10)})
 sns.countplot(x = 'University Rating', hue = 'Admit', hue_order = ['low','normal','high'], data = data)
 # Display the relationship between application documents and admission probability
 sns.set(rc = {'figure.figsize':(20, 10)})
 sns.countplot(x ='SOP', hue = 'Admit', hue_order = ['low','normal','high'], data = data)
 # Display the relationship between CGPA and admission probability
 sns.set(rc = {'figure.figsize':(20,10)})
 sns.boxplot(x = 'Admit', y = 'CGPA', data = data, order = ['low','normal','high'])
 Show the Relationship between Scientific Research Experience and Admission Probability
 sns.set(rc = {'figure.figsize':(20,10)})
 sns.boxplot(x = 'Admit', y = 'Research', data = data, order = ['low','normal','high'])
```

The following is a presentation and explanation of the output:

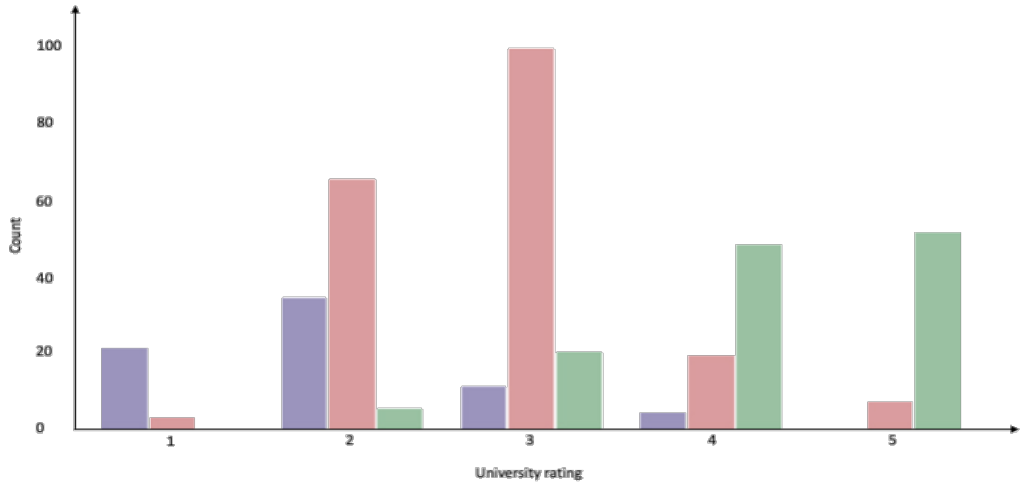
	Chance of Admit	GRE Score	TOEFL Score	CGPA	LOR	SOP	Research
Chance of Admit	1.000000	0.802610	0.791594	0.873289	0.669889	0.675732	0.553202
GRE Score	0.802610	1.000000	0.835977	0.833060	0.557555	0.612831	0.580391
TOEFL Score	0.791594	0.835977	1.000000	0.828417	0.567721	0.657981	0.489858
CGPA	0.873289	0.833060	0.828417	1.000000	0.670211	0.718144	0.521654
LOR	0.669889	0.557555	0.567721	0.670211	1.000000	0.729593	0.396859
SOP	0.675732	0.612831	0.657981	0.718144	0.729593	1.000000	0.444029
Research	0.553202	0.580391	0.489858	0.521654	0.396859	0.444029	1.000000

The correlation matrix shows us the correlation coefficients between seven features and tags, with core courses GPA, GRE, and TOEFL scores having a higher correlation with admissions probability, followed by application papers, letters of recommendation, and scientific research experiences having a lower correlation with admissions probability.

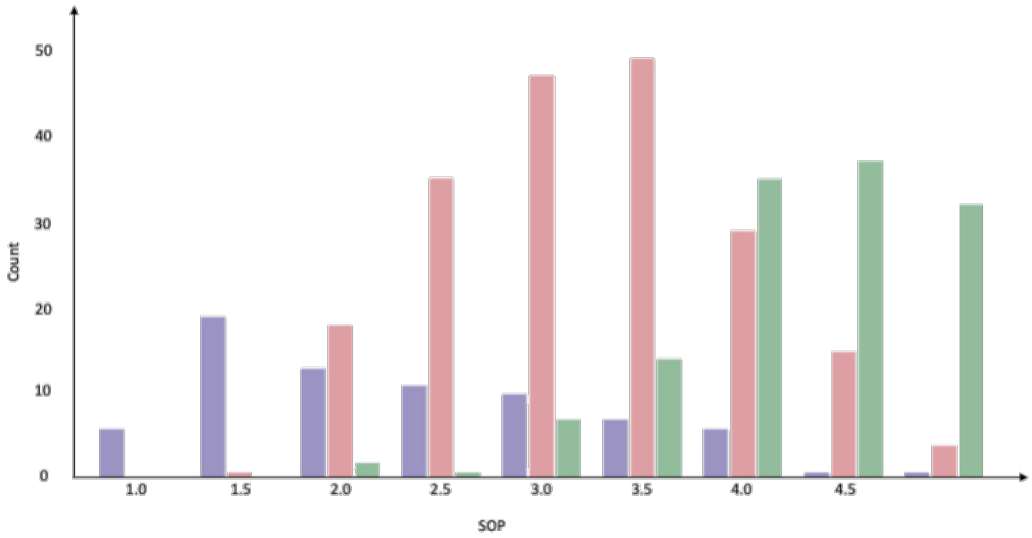
We divided Chance of Admit into three groups, with a low admission rate "low" of 0.6 or less, a medium admission rate "normal" of 0.6 to 0.8, and a high admission rate "high" of 0.8 or more. In the chart below, purple indicates a low admission rate, pink column indicates a medium admission rate, and green indicates a high admission rate.



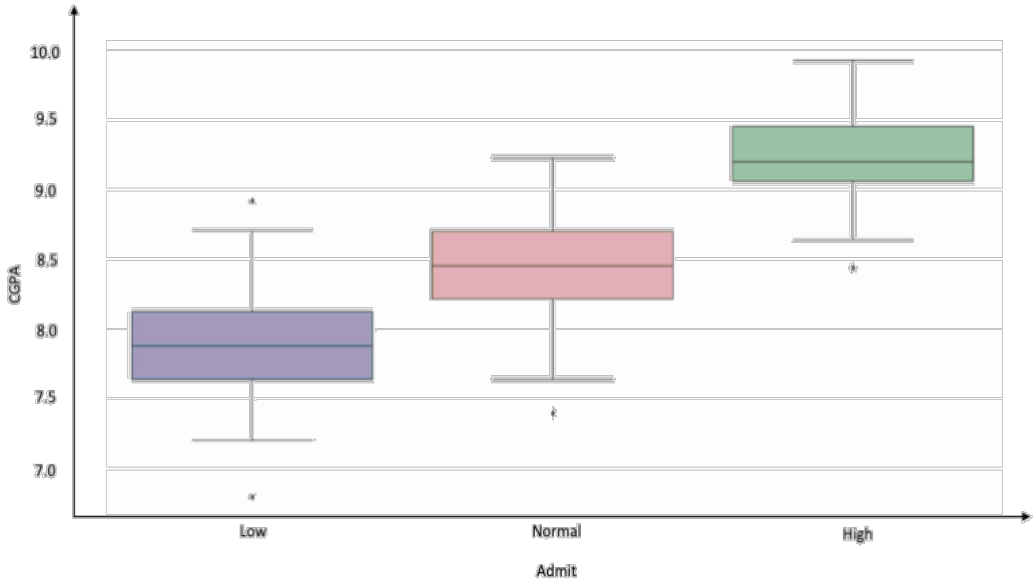
This diagram shows the relationship between GRE score and the admission probability. The abscissa is the admission probability admit, and the ordinate is the GRE score. It can be seen that the admission probability of the lower GRE score is also lower, and the higher GRE score is also higher, but this is not absolute, which indicates that there are other factors affecting the admission result.



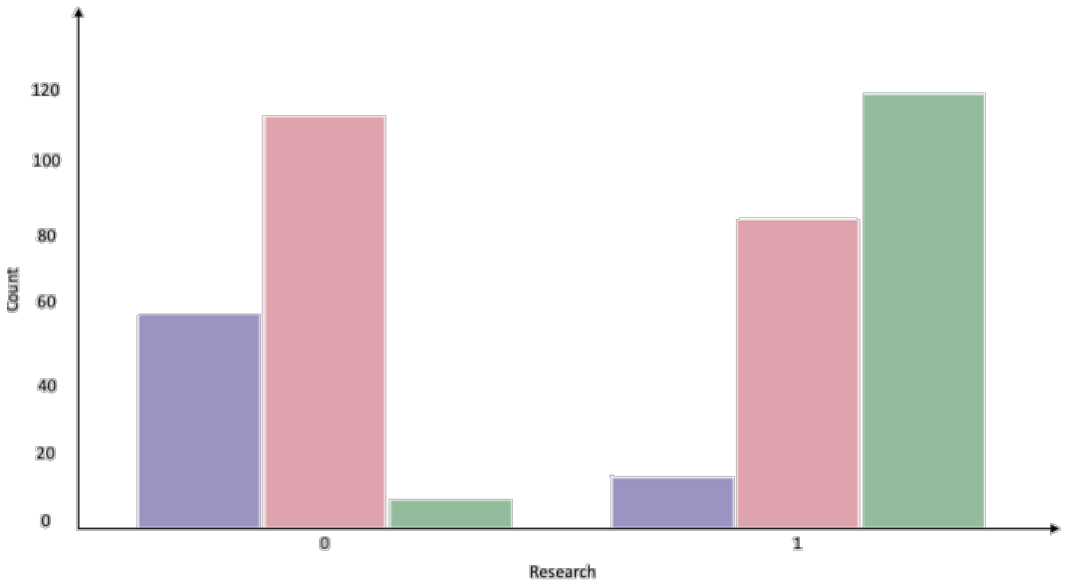
This picture shows us the relationship between the ranking of undergraduate schools and the admission probability. The abscissa is the ranking of undergraduate universities, and the ordinate is the number of students. We can see that the high admission rate (green) is mainly concentrated in the ranking scores of undergraduate schools of 4 and 5, that is, the top universities of undergraduate schools, so the ranking of undergraduate schools is also very important to the admission probability.



This chart shows the relationship between the application document and the admission probability. The abscissa is the score of the application document, and the ordinate is the number of students. It can be seen that the high admission rate (green) is mainly concentrated in 4-5, indicating that writing a quality application document plays a great role in the admission probability.



This image shows the relationship between CGPA and admission probability. The abscissa is the admission probability, and the ordinate is the CGPA score. There is no doubt that the university professors prefer the better students during the undergraduate course.



This picture shows the relationship between scientific research experience and admission probability. The abscissa indicates whether there is scientific research experience, and the ordinate is the number of students. It can be seen that the high admission rate (green) is mainly concentrated in 1, that is, among the students with scientific research experience. Among the students without scientific research experience, there are some students with high admission rate, but far less than the number of students with scientific research experience and high admission rate. So we think the scientific experience is a plus that helps improve admission.

From the above analysis, we found that the core courses GPA, GRE and TOEFL scores are the most representative and the most explanatory, and the correlation coefficient is very high, which plays a vital role in the admission results.

2 Model selection

Over the course of three months, we've been learning a lot from basic k nearest neighbor models to linear regression models that we used to be familiar with, to relatively complex tree models. Finding models that fit our data from existing models is the first step to think about before machine learning. We determine by calculating the mean square error of each model on our group of data.

The mean square error can be used to detect the deviation between the predicted value and the real value of the model. Specifically for the mse algorithm, the code is as follows:

```
import os
os.chdir('C:/temp')
# Library required to load this project
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics
# Load Dataset
data = pd.read_csv('uni.csv')
rate = data['Chance of Admit']
features = data.drop('Chance of Admit', axis = 1)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(features,rate,random_state=0)
from sklearn.neighbors import KNeighborsRegressor
reg=KNeighborsRegressor(n_neighbors=3)
```

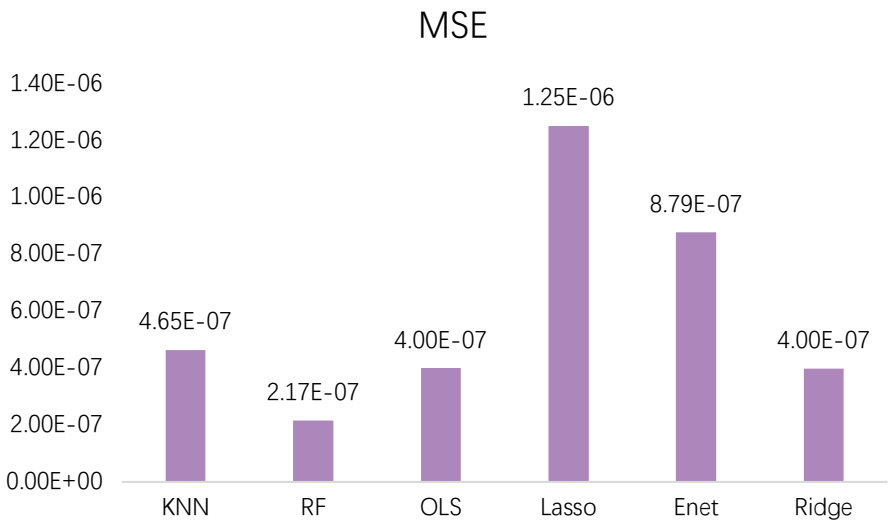
Import the required libraries and our data first, dividing the datasets into test sets and training sets. The mse is then calculated using the metrics provided in the sklearn.

The idea of calculation is: Firstly, different regression models are introduced to make them learn the training

set divided before; Then using the learning result to predict the tags of all samples in the data set, the predicted value is denoted as predicted; The mean square error is obtained by calculating the deviation of predicted from the original dependent variable value in the data by metrics. Finally the command outputs the mean square error.

The code is as follows:

```
Def RF X_train, X_test, features, rate: # Random forest
    from sklearn.ensemble import RandomForestRegressor
    model= RandomForestRegressor()
    model.fit(X_train, y_train)
    predicted= model.predict(features)
    mse = metrics.mean_squared_error(rate,predicted)
    return (mse/10000)
print('RF mse: ',RF(X_train, X_test, features, rate))

Linear regression of def LR: # X_train, X_test, features, rate
    from sklearn.linear_model import LinearRegression
    LR = LinearRegression()
    LR.fit(X_train, y_train)
    predicted = LR.predict(features)
    mse = metrics.mean_squared_error(rate,predicted)
    return (mse/10000)
print('LR mse: ',LR(X_train, X_test, features, rate))

Regression of: # Elastic Networks with def ENet X_train, X_test, features, rate
    from sklearn.linear_model import ElasticNet
    ENet=ElasticNet()
    ENet.fit(X_train, y_train)
    predicted = ENet.predict(features)
    mse = metrics.mean_squared_error(rate,predicted)
    return (mse/10000)
print('ENet mse: ',ENet(X_train, X_test, features, rate))

Def KN X_train, X_test, features, rate: #k Nearest Neighbor Model
    from sklearn.neighbors import KNeighborsRegressor
    KN =KNeighborsRegressor()
    KN.fit(X_train, y_train)
    predicted = KN.predict(features)
    mse = metrics.mean_squared_error(rate,predicted)
    return (mse/10000)
print('KN mse: ',KN(X_train, X_test, features, rate))

: #lasso regression of def lasso X_train, X_test, features, rate
    from sklearn.linear_model import Lasso
    lasso =Lasso()
    lasso.fit(X_train, y_train)
    predicted = lasso.predict(X_test)
    mse = metrics.mean_squared_error(y_test,predicted)
    return (mse/10000)
print('lasso mse: ',lasso(X_train, X_test, y_train, y_test))

Def ridge: #Ridge regression of X_train, X_test, features, rate
    from sklearn.linear_model import Ridge
    ridge =Ridge()
    ridge.fit(X_train, y_train)
    predicted = ridge.predict(features)
    mse = metrics.mean_squared_error(rate,predicted)
    return (mse/10000)
print('ridge mse: ',ridge(X_train, X_test, features, rate))
```



The mean square error of each model in this data

The mean square error values obtained by using multiple models in the same calculation are summarized in the above table. We can see that the magnitude of the mean square error of all models except Lasso regression is at the level of 10-7. The mse of stochastic forest model is the smallest, and the fitting degree should be the best, followed by K nearest neighbor model, linear regression model and ridge regression model. According to this, we will mainly rely on the above models.

3 K Nearest Neighbor Model and Its Optimization

3.1 K Nearest Neighbor Model with Default Parameters

Model We chose to practice from the simplest K-nearest neighbor model. The k nearest neighbor model with three nearest neighbors is used to study and predict our data. The code is as follows:

```
from sklearn.neighbors import KNeighborsRegressor
reg=KNeighborsRegressor(n_neighbors=3)
reg.fit(X_train,y_train)
reg.score(X_train,y_train)
0.8502719447259356
reg.score(X_test,y_test)
0.5664126838233539
```

The training set score was 0.85 and the training set score was 0.56. The results show that the overfitting problem exists and the generalization ability of the model is not good, but the scores of the training set and the test set need to be improved. So we try to optimize the k nearest neighbor model to improve the scores of the two terms.

Optimization of 3.2 K Nearest Neighbor Model

The three parts of the k nearest neighbor model that can be optimized are: The number of nearest neighbors is k, and the voting right of each nearest neighbor is equal weight or weighted by distance.

First of all, the optimal number of nearest neighbors and the voting mechanism with higher score are found out at one time. The value of k is selected between one and one hundred, and the voting mechanism is selected in equal weight uniform and weighted distance.

The code is as follows:

```
best_method= ''; best_k=-1; best_score=0.0
```

```
for method in ['uniform','distance']:
    for k in range(1,100):
        knn=KNeighborsRegressor(n_neighbors=k,weights=method)
        knn.fit(X_train,y_train);t=knn.score(X_test,y_test)
        if t>best_score:
            best_score=t;best_k=k;best_method=method
    print(best_k,best_score,best_method)
```

10 0.6319582885181687 uniform

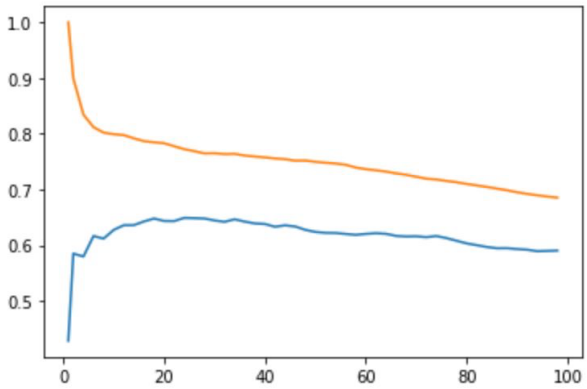
The results show that when the nearest neighbor number is 10, the score of the test set can reach 0. 63. This is an improvement from the initial default parameter model of 0.55, but it still needs to be improved.

We then tried to improve the model by using different distance measures. By default, the k nearest neighbor uses Euclidean distance, and we tried again using Manhattan distance.

The code is as follows:

```
testing_scores=[]
training_scores=[]
Ks=np.linspace(1,100,endpoint=False,dtype='int')
fig=plt.figure()
ax=fig.add_subplot(1,1,1)
for K in Ks:
        reg=KNeighborsRegressor(weights='uniform',n_neighbors=K,metric='manhattan')
        reg.fit(X_train,y_train)
        testing_scores.append(reg.score(X_test,y_test))
        training_scores.append(reg.score(X_train,y_train))
ax.plot(Ks,testing_scores,label="testing score")
ax.plot(Ks,training_scores,label="training score")
print(Ks[testing_scores.index(max(testing_scores))])
```

24



Finally, the optimal number of nearest neighbors is 24 and the graph above. The ordinate in the graph is the fraction of training set and test set, and the abscissa is the number of neighbors. The upper yellow line represents the trend of training set scores with the number of nearest neighbors, and the lower blue line represents the trend of test set scores with the number of nearest neighbors. The relationship between the number of nearest neighbors (k) and the scores of test sets and training sets under the measurement of Manhattan distance.

It can be seen from the graph that the scores of the test set and the training set will approach gradually, but the overfitting will always exist, the test set will stay between 60-70, and then there will be no room for increase.

3.3 Adjusted K Nearest Neighbor Model

The K-nearest neighbor model is tried again with the optimal parameters displayed in the optimization process.

```
reg=KNeighborsRegressor(weights='uniform',n_neighbors=24,metric='manhattan')
reg.fit(X_train,y_train)
```

0.7724533681987178
reg.score(X_train,y_train)
0.6492667193274698

The results show that the over-fitting problem is alleviated, but the scores of training set and test set are not high, and the over-fitting problem still exists.

4-Tree Model and Its Optimization

4.1 Comparison between decision tree model and stochastic forest model

The K-nearest neighbor model is not ideal, then we turn to the tree model to expect better results on the tree model.

We know that random forests outperform decision tree models in generalization and that a forest should perform better than a tree. And the mean square error of the stochastic forest model in the first mse calculation also stands out in the middle model. So we look forward to the performance of random forests.

We also compare the performance of decision tree model and stochastic forest model on the data before entering into stochastic forest.

The code is as follows:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.utils import shuffle
from sklearn.tree import DecisionTreeRegressor

data_shuffled = shuffle(data, random_state = 32)
num_of_samples = data_shuffled.shape[0]
split_line = int(num_of_samples * 0.75)
train_data = data.iloc[:split_line]
test_data = data.iloc[split_line:]

features = ['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA','Research']
trainx = train_data[features]
trainy = train_data['Chance of Admit']
testx = test_data[features]
testy = test_data['Chance of Admit']
model = RandomForestRegressor()

dt = DecisionTreeRegressor(random_state=0)
rf = RandomForestRegressor(random_state=0)
dt = dt.fit(trainx,trainy)
rf = rf.fit(trainx,trainy)
score_dt = dt.score(testx,testy)
score_rf = rf.score(testx,testy)
print("Single Tree:{}".format(score_dt),"Random Forest:{}".format(score_rf))
```

Single Tree:0.5502209491646172
Random Forest:0.7418181034885853

Before learning the decision tree model and the random forest model, we choose shuffle to redisrupt the data and reclassify the test set and training set, in order to avoid the inherent rule of the data arrangement and make the data set not random. Then use the new test set and training set to view the decision tree and random forest performance on the test set. The results showed that the score of single family tree model test set was 0.55, and that of random forest training set was 0.74. This is consistent with our perception that stochastic forest models do perform better in generalization.

4.2 Stochastic forest model with default parameters

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(trainx,trainy)
model.score(trainx,trainy)
0.9588845276465195
model.score(testx,texty)
0.7355492521457891
```

Under the default parameter, the result of random forest model shows that the result of training set is 0.95, the result of testing set is 0.73. Although the problem of overfitting still exists, the results of both results of K-nearest neighbor model have been greatly improved compared with those of K-nearest neighbor model.

4.3 Optimization of stochastic forest model

To improve the test set, we are trying to optimize random forests. In the process of optimization, some difficulties have been encountered: First, there are many parameters of random forest, besides the maximum number of features, the number of trees and the maximum depth of single family tree, there are also the maximum number of leaf nodes, the minimum sample leaf size and other parameters; The second is that random forests take more time to execute than other models, relatively testing the performance of computers.

Considering the characteristics of our data, there are only seven features and 400 samples. Then most of the random forest parameters will not have much impact, such as the maximum depth of the decision tree and so on. So from the multiple parameters I chose to try to find the number of optimal trees.

Here, the gridsearch provided by sklearn is used. The meaning of the gridsearch is to automatically adjust the parameters. As long as the parameters are input, the optimal results and parameters can be given. This is a method for small data sets.

The optimization process code is as follows:

```
from sklearn.model_selection import GridSearchCV
param_grid = {'n_estimators':np.arange(10,161,10)}
rf= RandomForestRegressor()
GS = GridSearchCV(rf,param_grid)
GS.fit(trainx,trainy)
print(GS.best_params_)
print(GS.best_score_)
{'n_estimators': 130}
0.7152888078965246
model1 = RandomForestRegressor(n_estimators=GS.best_params_['n_estimators'])
model1.fit(trainx,trainy)
prediction=model1.predict(testx)
print(model1.score(trainx,trainy))
print(model1.score(testx,testy))
0.966017031135048
0.7634343912408825
```

By default, the number of trees in a random forest is 100, and the optimization rule for the number of trees is "as many as time and memory allow." After trying a lot of groups, we found that the final results were similar. The result shown above is the result obtained by finding the optimal number in the 10-161 range of the default value 100.

In fact, the output result of the program will be different every time it runs, but python can save the optimal parameters to the dictionary directly, and then can introduce the optimal number from the dictionary in the process of using the model, and then predict it. This ensures that every n brought in will be guaranteed.

From the final results, it can be found that the test set score obtained by using the number of optimal trees increased by 3 points, which is not much better than that obtained by using the default parameters. But looking back to the k nearest neighbor model, the stochastic forest is still better.

Therefore, we think that the stochastic forest model itself has been relatively perfect, and the subsequent optimization process can not greatly improve its training set.

5. Four Regression Models and Their Optimization

Before the four models are used for learning and prediction, the data sets of training models need to be disturbed by shuffle before the training sets and test sets are segmented. Because the raw data may be arranged in a certain order, the data may be arranged randomly after being disrupted. Using data from shuffle disruptions can improve model quality, improve predictive performance, and prevent overfitting.

The code is as follows:

```
from sklearn.utils import shuffle
features1, rate1=shuffle(features,rate)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(features1,rate1,random_state=0)
```

5.1 Linear regression model

5.1.1 Linear Regression Model with Default Parameters

Among the four linear regression prediction models, learning and prediction are carried out from the most commonly used OLS linear regression.

The code is as follows:

```
# Reference linear regression prediction model
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
# Coefficient of each feature
lr.coef_
array([0.00161322,0.0033721,0.00424734,-0.0071272,0.02330074,0.120739
31,0.02860655])
Learning Effect of # Training Set
lr.score(X_train, y_train)
0.8024381943162312
Learning Effect of # Test Set
lr.score(X_test, y_test)
0.8030781977598971
```

There is no overfitting problem at this time, but the scores of the training set and the test set are not very high, so the regularization method is then used to "punish" the characteristic variable coefficients, and the L1regularization (Lasso regression) and L2regularization (Ridge regression) methods are used to optimize the model.

5.2 Lasso regression model

5.2.1 Lasso Regression Model with Default Parameters

The regression prediction is first performed using the L1regularization method (Lasso regression) with the regularization parameter alpha value of one by default.

The code is as follows:

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=1.0,max_iter=1000000)
```

```
lasso.fit(X_train, y_train)
Learning Effect of # Training Set
lasso.score(X_train, y_train)
0.26374400988129165
Learning Effect of # Test Set
lasso.score(X_test, y_test)
0.24485683610477374
# View number of reserved features (coefficient is non-zero): 1 (too few reserved features)
import numpy as np
np.sum(lasso.coef_ != 0)
1
```

It can be seen that when alpha is the default value of one, the learning effect is very poor, the training set score is only about 26.37%, the test set score is only about 24.48%, and the number of reserved features is only one.

5.2.2 Optimization of Lasso Regression Model

Because the learning effect of Lasso regression model is poor under the default alpha value, we look for the best alpha value, expand the heuristic range of alpha value, and hope to find the best parameters.

First select the alpha value, and the code is as follows:

```
from sklearn.linear_model import LassoCV
import numpy as np
alphalist=10**np.linspace(-4,4,100)
lassocv=LassoCV(alphas=alphalist,max_iter=1000000)
lassocv.fit(X_train, y_train)
# Value of best alpha
lassocv.alpha
0.00021049041445120198

# View number of reserved features (factor is non-zero): 7
np.sum(lassocv.coef_ != 0)
7
Learning Effect of # Training Set
lassocv.score(X_train, y_train)
0.8023747303589627
Learning Effect of # Test Set
lassocv.score(X_test, y_test)
0.8037739786799406
```

The first round of improvement is then carried out with the following code:

```
alist1=list(alphalist)
pos1=alist1.index(lassocv.alpha_)
alphalist1=np.linspace(alist1[pos1-1],alist1[pos1+1],100)
lassocv1=LassoCV(alphas=alphalist1)
lassocv1.fit(X_train, y_train)
# Best alpha
lassocv1.alpha
0.00023125421643645762
lassocv1.score(X_train, y_train)
0.8023615882000796
lassocv1.score(X_test, y_test)
0.8038357149042904
```

Perform a second round of improvement with the following code:

```
alist2=list(alphalist1)
pos2=alist2.index(lassocv1.alpha_)
alphalist2=np.linspace(alist2[pos2-1],alist2[pos2+1],100)
```

lassocv2=LassoCV(alphas=alphalist2)
lassocv2.fit(X_train, y_train)
*lassocv2.alpha_*
0.00023188924399918624
lassocv2.score(X_train, y_train)
0.8023611667928228
lassocv2.score(X_test, y_test)
0.8038375833472293
np.sum(lassocv2.coef_ != 0)
7
*lassocv2.coef_*
array([0.00171601,0.00342178,0.00396803,-0.0056037,0.02269404,0.11777148, 0.02723177])

After the second round of improvement, the four decimal places of the training set and test set scores have not changed, so the improvement can be stopped. It can be seen that the training set score and the test set score of the optimized Lasso regression model are much higher than those of the default parameter, the training set score reaches about 80.23%, the test set score reaches about 80.38%, the alpha value is about 0.0002319, and all features are reserved.

5.3 Ridge regression model

5.3.1 Ridge Regression Model with Default Parameters

The regression prediction is then performed using the L2regularization method (Ridge regression) with the regularization parameter alpha value of one by default.

The code is as follows:

from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
Learning Effect of # Training Set
ridge.score(X_train, y_train)
0.8023275869049209
Learning Effect of # Test Set
ridge.score(X_test, y_test)
0.8034349361534416
 # View number of reserved features (factor is non-zero): 7
np.sum(ridge.coef_ != 0)
7

The overfitting problem does not exist under the default parameters, but the score is fair. Then the search range of alpha value is enlarged, the optimal alpha value is found, and the model is optimized.

5.3.2 Optimization of Ridge Regression Model

First select the alpha value, and the code is as follows:

from sklearn.linear_model import RidgeCV
import numpy as np
*alphalist=10**np.linspace(-3,3,100)*
ridgecv=RidgeCV(alphas=alphalist)
ridgecv.fit(X_train, y_train)
 # Value of best alpha
ridgecv.alpha
0.7054802310718645
Learning Effect of # Training Set

ridgecv.score(X_train, y_train)
0.8023815775346189
Learning Effect of # Test Set
ridgecv.score(X_test, y_test)
0.8033560815352233

The first round of improvement is then carried out with the following code:
alist1=list(alphalist)
pos1=alist1.index(ridgecv.alpha_)
alphalist1=np.linspace(alist1[pos1-1],alist1[pos1+1],100)
ridgecv1=RidgeCV(alphas=alphalist1)
ridgecv1.fit(X_train, y_train)
 # Best alpha
*ridgecv1.alpha_*
0.7333120021585113
ridgecv1.score(X_train, y_train)
0.8023771854079855
ridgecv1.score(X_test, y_test)
0.8033644080111203

Perform a second round of improvement with the following code:
alist2=list(alphalist1)
pos2=alist2.index(ridgecv1.alpha_)
alphalist2=np.linspace(alist2[pos2-1],alist2[pos2+1],100)
ridgecv2=RidgeCV(alphas=alphalist2)
ridgecv2.fit(X_train, y_train)
 # Best alpha
*ridgecv2.alpha_*
0.7340980509325636
ridgecv2.score(X_train, y_train)
0.8023770591529804
ridgecv2.score(X_test, y_test)
0.803364640459238
 # View number of reserved features (factor is non-zero): 7
np.sum(ridgecv2.coef_ != 0)
7
 # Coefficient of each feature
*ridgecv2.coef_*
array([0.00169937, 0.00348367,0.00457518,-0.00665175,0.02345317,0.11654077,0.0282724])

After the second round of improvement, the four decimal places of the training set and test set scores have not changed, so the improvement can be stopped. In this case, the training set score is about 80.24%, the test set score is about 80.34%, and the optimal alpha value is about 0.734.

5.4 ElasticNet regression model

ElasticNet (elastic network model) combines Ridge regression and Lasso regression in theory, which can adjust the parameters alpha and l1_ratio of Ridge regression and Lasso regression simultaneously. Therefore, it combines the common advantages of Ridge regression and Lasso regression (stability of Ridge regression, iteration convergence of Lasso regression and feature number screening), and can effectively deal with multiple collinearity problems.

5.4.1 ElasticNet Regression Model with Default Parameters

First perform regression learning under the default parameters, default alpha=1.0, l1_ratio=0.5.
The code is as follows:

```
from sklearn.linear_model import ElasticNet
ENet=ElasticNet(alpha=1.0,l1_ratio=0.5,max_iter=1000000)
ENet.fit(X_train, y_train)
ENet.score(X_train, y_train)
0.546039172867677
ENet.score(X_test, y_test)
0.5269778580299556
np.sum(ENet.coef_ != 0)
1
```

In this case, the scores of the training set and the test set are low, and only one feature is reserved. Therefore, the parameters are changed to make alpha=0.01 and l1_ratio=0.1 to see if there is improvement.
The code is as follows:

```
from sklearn.linear_model import ElasticNet
ENet=ElasticNet(alpha=0.01,l1_ratio=0.1, max_iter=10000)
ENet.fit(X_train, y_train)
ENet.score(X_train, y_train)
0.7993625621323148
ENet.score(X_test, y_test)
0.8041262108011181
```

At this time, the training set score and the test set score are greatly improved, and then the search range of the parameters is enlarged, and the optimal parameters are expected to be found, so that the learning effect is the best.

5.4.2 Optimization of ElasticNet Regression Model

Look for the best parameters alpha and l1_ratio with the following code:

```
from sklearn.linear_model import ElasticNetCV
import numpy as np
alphalist=10**np.linspace(-3,3,100)
l1list=np.linspace(0.01,1.0,100)
ENet=ElasticNetCV(alphas=alphalist,l1_ratio=l1list,max_iter=100000)
ENet.fit(X_train, y_train)
ENet.alpha_ , ENet.l1_ratio_
(0.005336699231206312, 0.01)
ENet.score(X_train, y_train)
0.802126095039597
ENet.score(X_test, y_test)
0.803668317568403
np.sum(ENet.coef_ != 0)
7
```

It can be seen that the training set score and the test set score are greatly improved compared with the default parameter. The training set score is about 80.21%, the test set score is about 80.37%, the optimal alpha is 0.0053, and the optimal l1_ratio is 0.01.

6 Summary and Significance

6.1 Summary of Six Models

Among the above six models, the best results are as follows:

	K nearest neighbor model	Random forest	Linear regression	Lasso regression	Ridge regression	ElasticNet regression
Training set score	0.772453	0.965798				
Test Set Score	0.649267	0.763266				

It can be seen that the training set and test set scores of stochastic forest model are the highest, the training set score is about 96.58%, the test set score is about 76.33%.

6.2 Individual hypothesis prediction

Assume that the condition background of a classmate is shown in the following figure:

GRE Score	TOFEL Score	University Rating	SOP	LOR	CGPA	Research
325	110	5	4	4	9.25	0

It can be seen that the GRE score of this classmate is 325, the TOEFL score is 110, the relative ranking score of undergraduate university is five, the individual statement (SOP) score is four, the recommendation letter (LOR) score is four, the CGPA ten-point system score is 9.25 - that is, 3.7 points when the full score is 4.0, and there is no scientific research experience. It can be seen that this is a relatively good background conditions of the students, using the stochastic forest model to predict the admission rate of the students.

The code is as follows:

```
pre_data=pd.read_excel('predict.xlsx')
features=['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA','Research']
prediction=model.predict(pre_data[features])
print(prediction)
```

0.904

The random forest model was used to predict that the admission rate of this student was 90.4%, so it was predicted that if the student applied for the American university of TOP50, the admission rate would be about 90.4%, which was quite high.

6.3 Research significance

The study, based on machine learning's prediction of the admission rate of American research, aims to provide reference for students who wish to apply under the conditions of known personal background and better understand their own situation. Because the selected data sample is the applicant situation of a university in TOP50 in the United States, through the regression results of machine learning, we can preliminarily understand the requirements of the high-level universities in the United States for undergraduate applicants, the importance of personal comprehensive quality, not only require good learning results, language skills, but also require strong social practice experience and so on. The prediction results can provide reference for students who wish to apply, have a more objective and fair understanding of the institutions and themselves, and better choose the appropriate schools in the application process.