FTE4560 Project1

--GroupMember--

--119020062 Wenzhong Xu--

--119020084 Yipeng Zou--

--119020286 Ziqi Liao--

FTE4560 Project1

--GroupMember--

--119020062 Wenzhong Xu--

--119020084 Yipeng Zou--

--119020286 Ziqi Liao--

1 Data Source and Data Description

Our data is about polish bankruptcies. Here are the data sources.

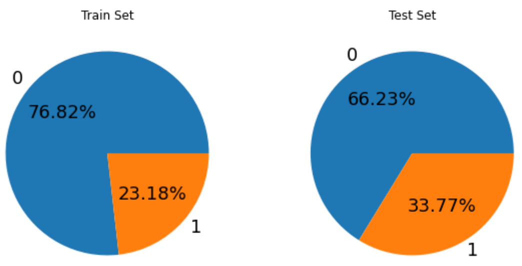
Polish companies bankruptcy data Data Set

We used only a subset of the total data. We have 64 features, All features are continuous . And its labels are 0 and 1, with 1 representing bankrupt and 0 representing unbankrupt. Data has been shred into training datasets (949 samples) and test datasets (151 samples).

2 Data Exploring

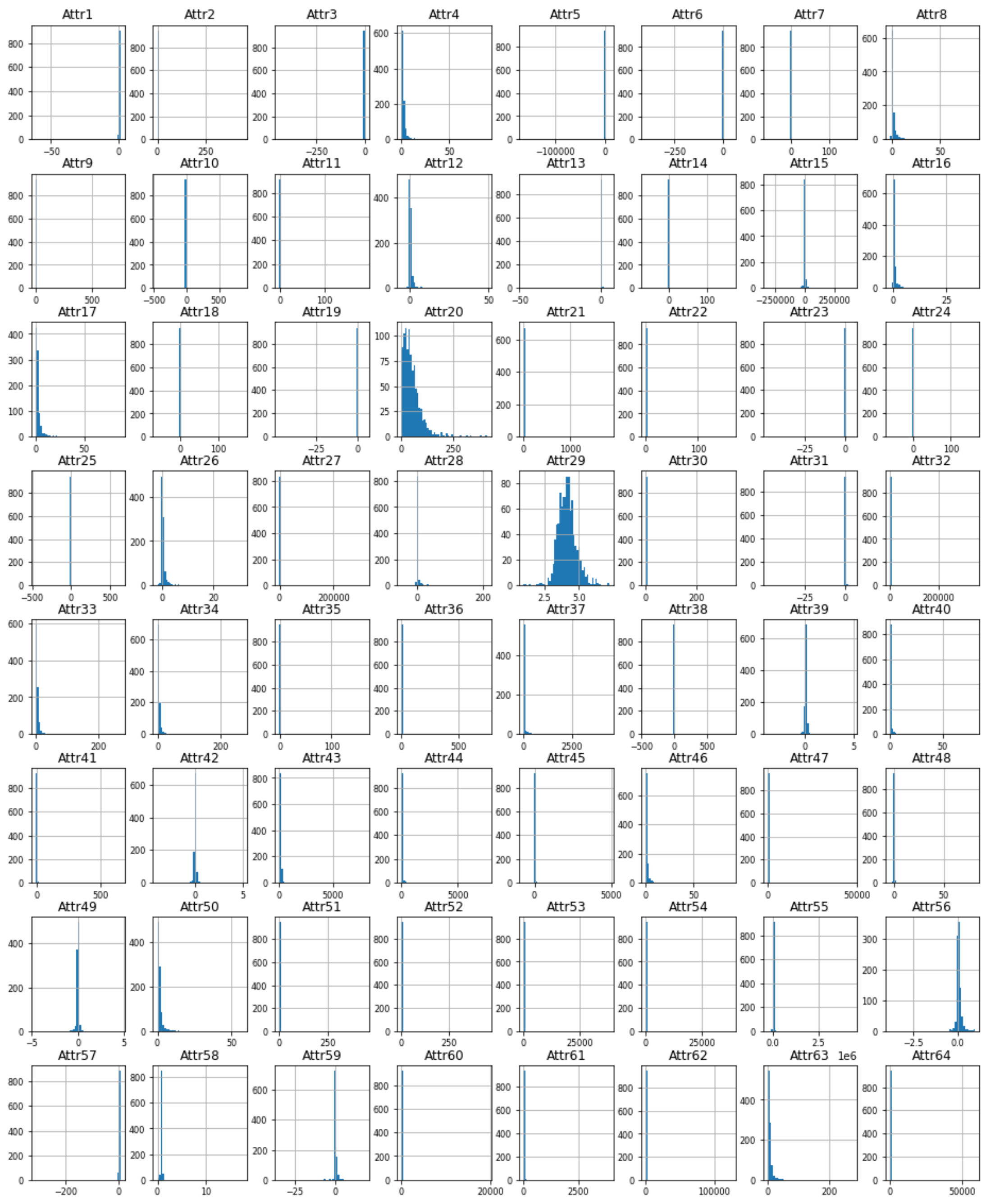
2.1 Unbalanced data

The pie chart shows that the data is very uneven, and it is worth noting that the training set and test set are not evenly distributed.



2.2 Features Distribution

All our features are continuous, so we can make histogram to check the training set distribution of each feature, as shown below:



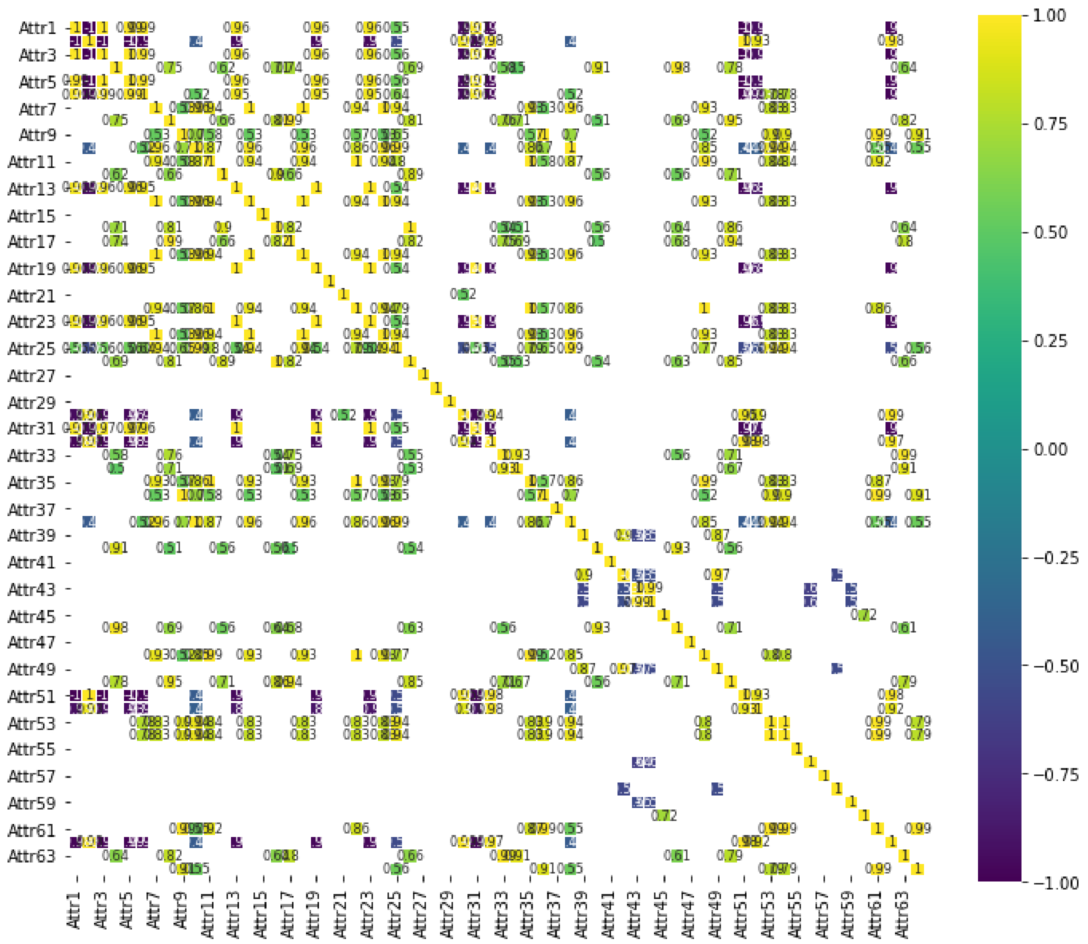
From the histogram, we can find some information:

- The distribution of most features is extremely uneven and there are a lot of outliers.
- The magnitude of the features are inconsistent. This needs to be standardized or normalized.

These problems will be solved in the pre-processing stage.

2.3 Correlation of features

The extreme correlation of features leads to multicollinearity, which reduces the effect of some models. We used a thermal map to check the correlation of features(training set):

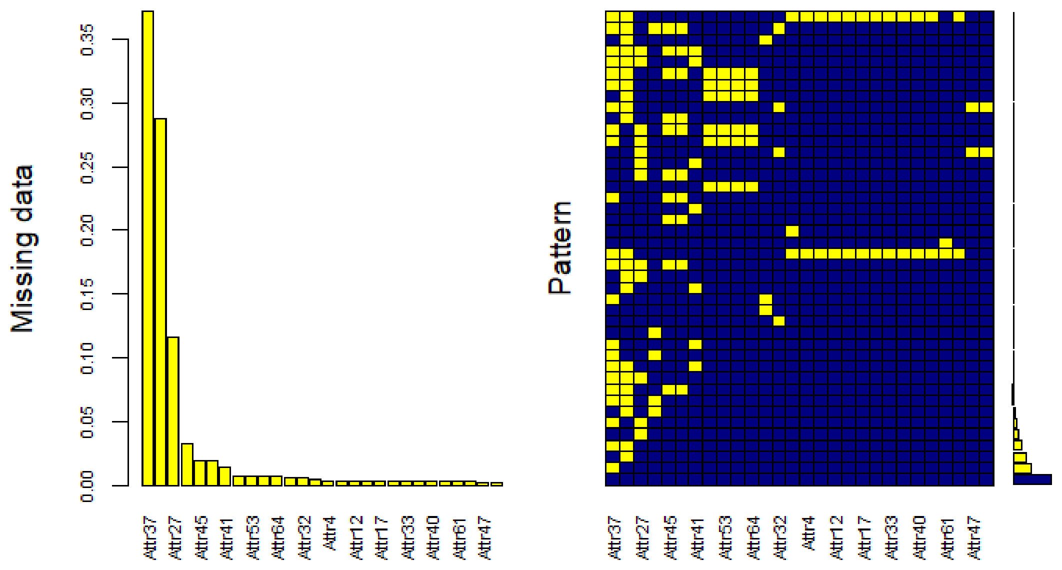


Only pairs with Pearson correlation coefficients greater than 0.7 (highly correlated) are shown here, and we can see that the data has serious correlation problems.

This will be resolved during data preprocessing.

2.4 Missing Values

In the given dataset, the missing values are represented as "?". Thus, we need to replace this symbol with `nan` so that the computer can recognize the missing values. After that, we count the number of the missing values for each parameter. Here are the results of the Training set.



From the above histogram(left), we found that most attributes (N=61) have less than 5% missing values. Only `Attr37`, `Attr27`, `Attr45` have more than 5% missing values.

The figure on the right shows the missing pattern. The yellow features in each row represent the patterns that these features are missing at the same time, and the corresponding histogram to the right of each row represents the proportion of this pattern in our data

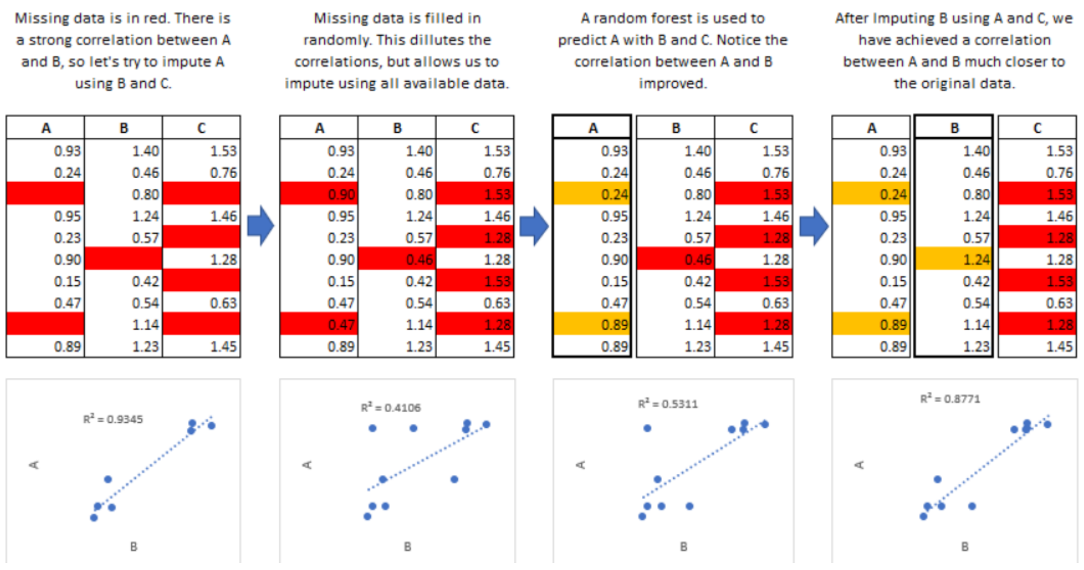
Therefore, the problem of missing values is also serious, and we will deal with the missing values in the pre-processing stage.

3 Data Preprocess

3.1 Handling Missing Values

The first step to clean the data is to handle the missing values. Due to the high proportion of partial features missing, it is no longer suitable to use median or average to fill our data. We tried **MICE** algorithm, which has the characteristics of universality and rapidity, and has a good effect for all kinds of missing value ratio.

The following are the core principles of the MICE algorithm:



Missing data is in red. There is a strong correlation between A and B, so let's try to impute A using B and C.

A	B	C
0.93	1.40	1.53
0.24	0.46	0.76
🟥	0.80	🟥
0.95	1.24	1.46
0.23	0.57	🟥
0.90	🟥	1.28
0.15	0.42	🟥
0.47	0.54	0.63
🟥	1.14	🟥
0.89	1.23	1.45

Missing data is filled in randomly. This dillutes the correlations, but allows us to impute using all available data.

A	B	C
0.93	1.40	1.53
0.24	0.46	0.76
0.90	0.80	1.53
0.95	1.24	1.46
0.23	0.57	1.28
0.90	0.46	1.28
0.15	0.42	1.53
0.47	0.54	0.63
0.47	1.14	1.28
0.89	1.23	1.45

A random forest is used to predict A with B and C. Notice the correlation between A and B improved.

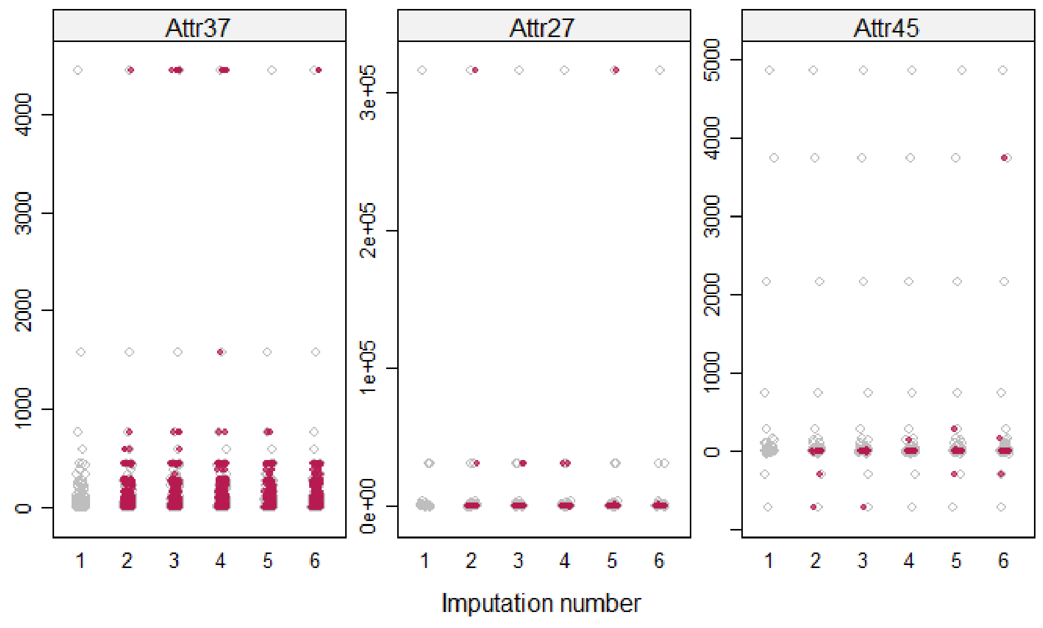
A	B	C
0.93	1.40	1.53
0.24	0.46	0.76
0.24	0.80	1.53
0.95	1.24	1.46
0.23	0.57	1.28
0.90	0.46	1.28
0.15	0.42	1.53
0.47	0.54	0.63
0.89	1.14	1.28
0.89	1.23	1.45

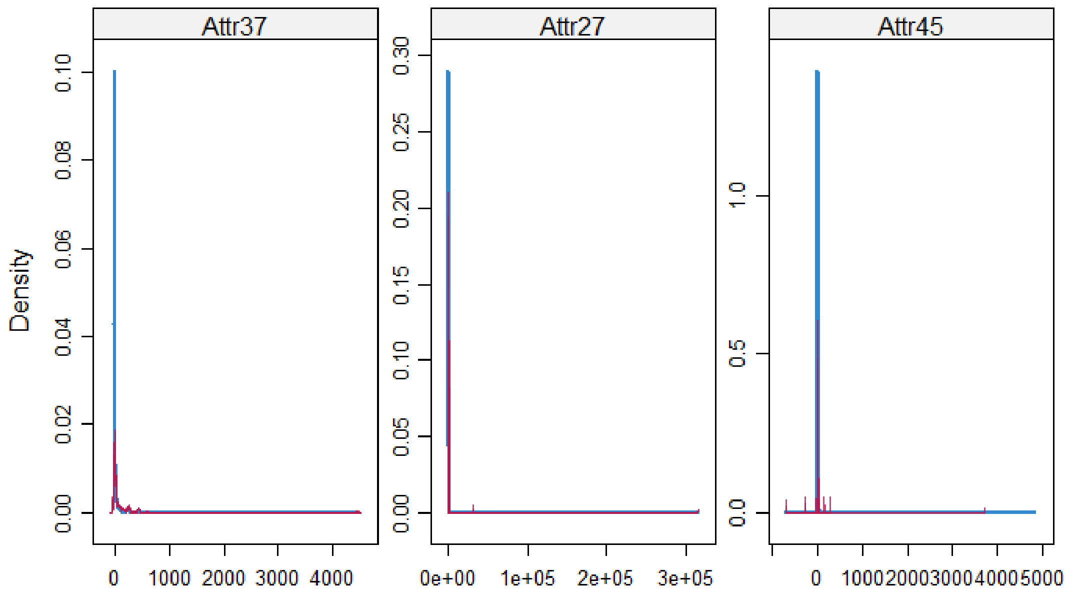
After Imputing B using A and C, we have achieved a correlation between A and B much closer to the original data.

A	B	C
0.93	1.40	1.53
0.24	0.46	0.76
0.24	0.80	1.53
0.95	1.24	1.46
0.23	0.57	1.28
0.90	1.24	1.28
0.15	0.42	1.53
0.47	0.54	0.63
0.89	1.14	1.28
0.89	1.23	1.45

This process continues until all specified variables are interpolated.

We use the MICE package in R to implement the algorithm on the feature data of training set. With these three features(`Attr37`, `Attr27`, `Attr45`) with the highest proportion, we can view the filling results(training set).





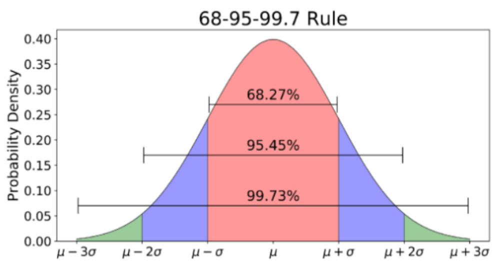
We can see filling results and perfect matching of raw data, which gives us more confidence in MICE algorithm.

Finally, we take the filling results in the training set as the real value, then combine the training set and test set, and fill the training set with the whole feature data.

3.2 Adjusting Data

3.2.1 Exclude Outliers

By the law of large number, we assume the values of our attributes follow the normal distribution. Thus, the range of the value of each attributes should be in $[\mu - 3\sigma,\ \mu + 3\sigma]$ by the **68-95-99.7 Rule**.



Thus, we will denote the data point as outlier if it is out of the range $[\mu - 3\sigma,\ \mu + 3\sigma]$, and set the value of the outlier as the threshold $\mu - 3\sigma$ or $\mu + 3\sigma$. The code to exclude outliers is:

```
def outsideAdj(self):
    mu = self.train_x.mean()
    sigma = np.sqrt(self.train_x.var())
    upper_bound = mu + 1 * sigma
    lower_bound = mu - 1 * sigma
    for attr in lower_bound.index:
        self.train_x[attr][self.train_x[attr] > upper_bound[attr]] =
upper_bound[attr]
        self.test_x[attr][self.test_x[attr] > upper_bound[attr]] =
upper_bound[attr]
        self.train_x[attr][self.train_x[attr] < lower_bound[attr]] =
lower_bound[attr]
        self.test_x[attr][self.test_x[attr] < lower_bound[attr]] =
lower_bound[attr]
    self.train_x = (self.train_x - self.train_x.min()) /
(self.train_x.max() - self.train_x.min())*10
    self.test_x = (self.test_x - self.test_x.min()) /
(self.test_x.max() - self.test_x.min())*10
```

3.2.2 Data Scale Down

In some models like `Least Square with Gaussian Basis` or `SoftMax`, we will do exponential transformation to the original data. If we just put the original data to do the transformation, we may encounter the problem that the transformation results are too large or too small. Therefore, we have to scale down the origin data to scale down it and reduce the variance. We will do two steps of the transformation:

1. Take square root of each data.
2. Do the Min-Max transformation.

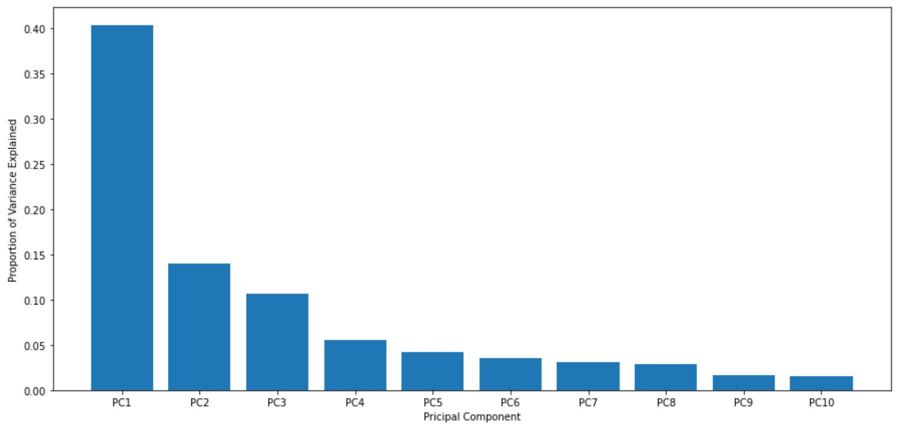
3.2.3 PCA Dimension Reduction

Due to the strong correlation of the data, we considered using Principal Component Analysis(PCA) to reduce the dimension of the data.

Principal Component Analysis (PCA) is a linear dimensionality reduction technique that can be utilized for extracting information from a high-dimensional space by projecting it into a lower-dimensional sub-space. It tries to preserve the essential parts that have more variation of the data and remove the non-essential parts with fewer variation.

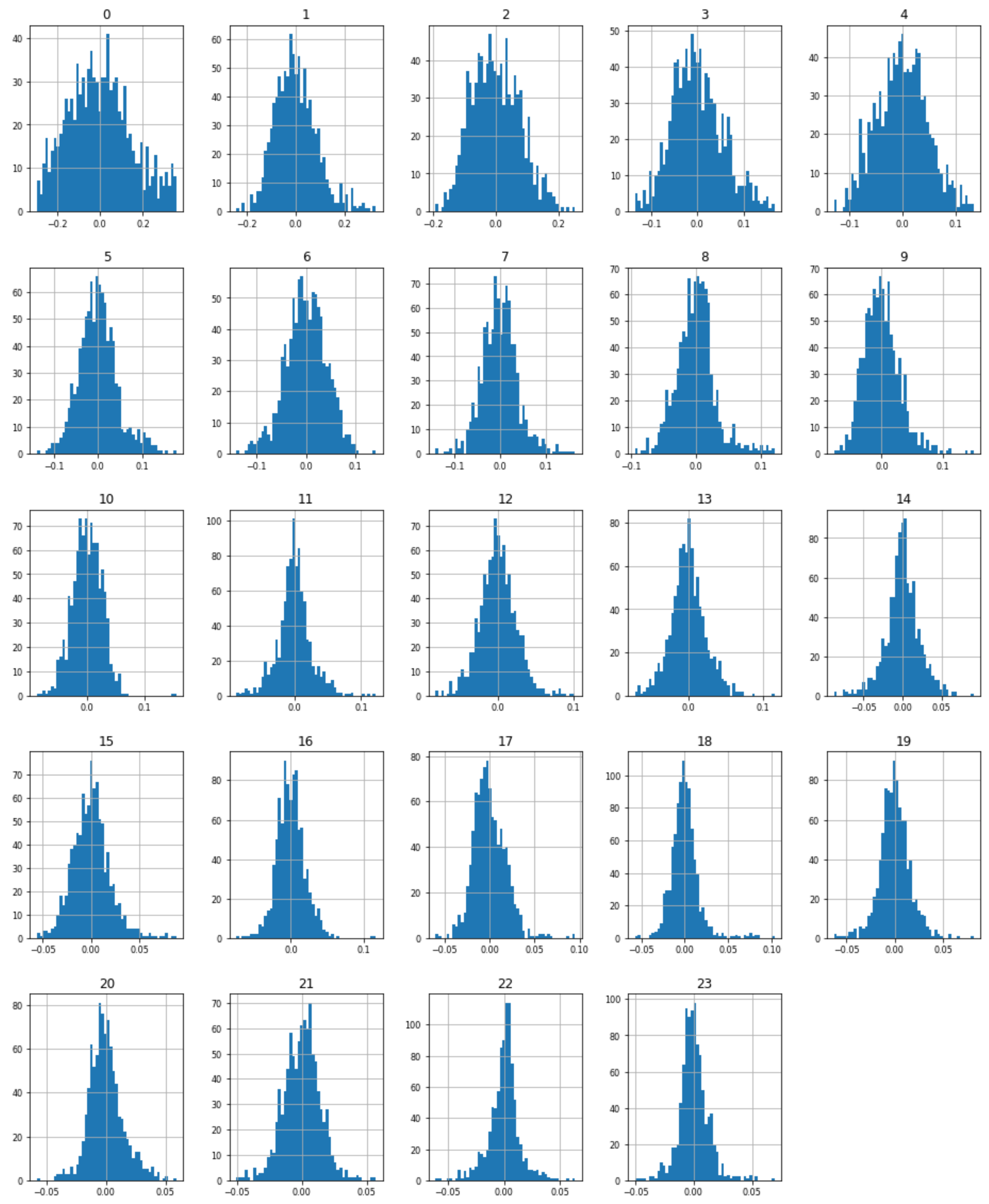
We implemented PCA using Python's Sklearn package, which preserved 97% of the information and ultimately reduced the dimensions of the 64-dimensional features to 24-dimensional.

The figure below shows the proportion of interpretation information corresponding to the top ten new features:



3.2.4 Result of preprocess

As shown in the figure below, after data preprocessing, we solved outlier, magnitude and correlation problems (because each feature generated by PCA has no correlation).



4 Model Summary

4.1 Performance Summary

We use weighted average to calculate accuracy, precision, recall and F1-score.

Model	Accuracy	Precision	Recall	AUC	F1-score	Training Time	Developer
KNN	75%	75%	75%	72.00%	0.65	NA	Ziqi Liao
LS	72%	76%	72%	79.67%	0.66	1.2s	Wenzhong Xu
LS Gaussian	79%	**80%**	73%	76.37%	0.73	8.6s	Yipeng Zou
LDA	73%	76%	75%	76.00%	0.76	0.002s	Ziqi Liao
Logistic	74%	79%	74%	79.61%	0.69	9.1s	Wenzhong Xu
Softmax	75%	76%	63%	74.31%	0.69	17m37s	Yipeng Zou
Tree	79%	78%	**79%**	81.01%	**0.78**	**0.06s**	Ziqi Liao
RF	**80%**	78%	77%	**87.17%**	**0.78**	12m11s	Yipeng Zou
NN	77%	77%	77%	80.37%	0.77	2.2s	Wenzhong Xu

5 Build Classification Model

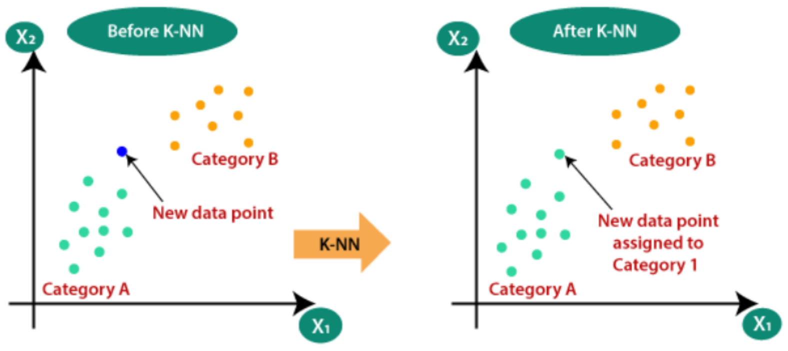
5.1 KNN Classifier Model

5.1.1 Overview of KNN Classifier Model

K-Nearest Neighbour(KNN) is one of the simplest Machine Learning algorithms based on Supervised Learning technique. It assumes the similarity between the new data and available data and put the new data into the class that is most similar to the available class.

When we build a KNN model, we need save the train features and train labels. It should be noted that, since Euclidean distance is calculated during KNN model prediction, our data need to be standardized. When we get a new data we need to do the following steps:

- **Step-1**: Select the number K of the neighbors and threshold.
- **Step-2**: Calculate the Euclidean distance between features of new data and all train features.
- **Step-3**: Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4**: Calculate the proportion of k samples that are class 2, which is score of class 2 for this data
- **Step-5**: If its score is greater than the threshold, we classify it as class 2

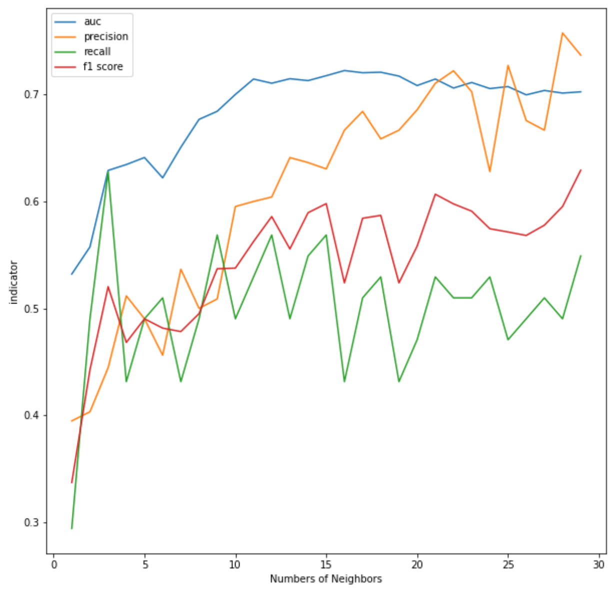


The implementation of KNN is not difficult. Here is our code:

```
class knn:
    def __init__(self,k):
        assert 1<=k
        self.k = k
        self.X = None
        self.y = None

    def fit(self,train_X,train_y):
        self.X = train_X
        self.y = train_y

    def predict(self,test_X):
        n = test_X.shape[0]
        result = np.ones(n)
        for i in range(n):
            new_X = test_X.iloc[i,:]
            nn = (np.argsort(((self.X - new_X)**2).sum(1)))[:self.k]
            result[i] = self.y.iloc[nn,].mean()
        return result
```

5.1.2 Result Analysis

Now, we use KNN in our pre-processed data. Due to unbalanced data, we take AUC, precision rate, recall rate and f1-score as the evaluation criteria. In our data, the following is the change of AUC, precision rate, recall rate and f1-score in test data when K goes from 1 to 30 (threshold = 0.3 for reasons that will be explained later).



We find that AUC reaches its maximum 0.7224 and precision rate, recall rate and f1-score were also close to the highest values when k is 16. So we chose k=16 as the parameter of the final model. The following is the result of KNN on the test set when K =16 threshold = 0.5.

Testing threshold = 0.5	Precision	Recall	F1-score	Support
Class 1	0.70	0.98	0.81	100
Class 2	0.80	0.16	0.26	51
Accuracy			0.70	151
Macro Avg	0.75	0.57	0.54	151
Weighted Avg	0.73	0.70	0.63	151
AUC			0.72	151

We found that the results were mediocre, especially for class 1, whose recall rate was low at 0.16, and whose AUC was relatively high, so we might be able to change the threshold to improve the predicted performance. The following is the result of KNN on the test set when K =16 threshold = 0.3.

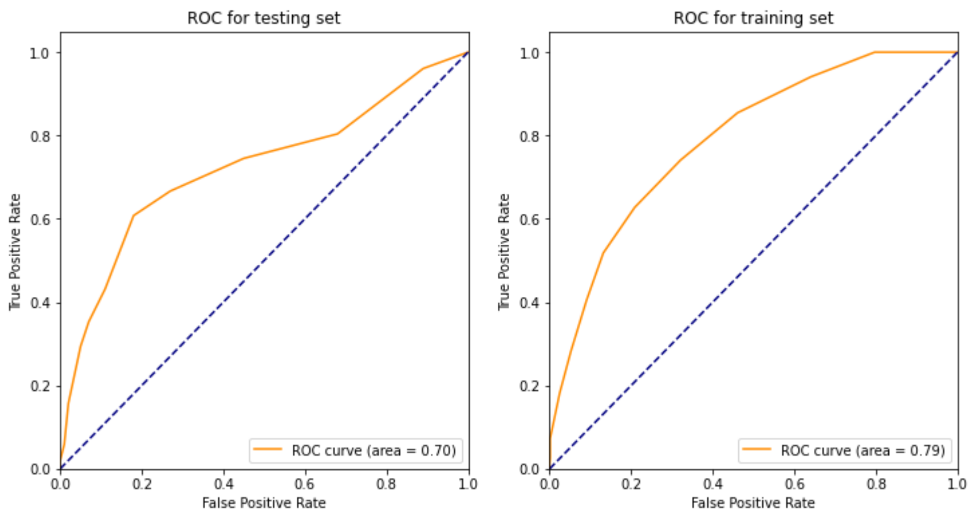
Testing threshold = 0.3	Precision	Recall	F1-score	Support
Class 1	0.80	0.82	0.81	100
Class 2	0.63	0.61	0.62	51
Accuracy			0.75	151
Macro Avg	0.72	0.71	0.72	151
Weighted Avg	0.75	0.75	0.65	151
AUC			0.72	151

It can be found that there is a great improvement in the model. In addition to the slightly reduced precision of Class 2 and recall of Class 1, other indicators, especially the recall of Class 2, have been greatly improved.

Let's look at the performance of the model on the training set when K =16 threshold = 0.3:

Training threshold = 0.3	Precision	Recall	F1-score	Support
Class 1	0.80	0.90	0.85	100
Class 2	0.74	0.55	0.63	51
Accuracy			0.78	151
Macro Avg	0.77	0.72	0.74	151
Weighted Avg	0.78	0.78	0.77	151
AUC			0.79	151

Here is the ROC curve:



In order to check the influence of different k on model performance, we also use k=1,3,5,10,15 as comparison values. Their results are as follows (a good threshold is selected for different k values).

K =1 threshold is invalid	Precision	Recall	F1-score	Support
Class 1	0.68	0.77	0.72	100
Class 2	0.39	0.29	0.34	51
Accuracy			0.61	151
Macro Avg	0.54	0.53	0.53	151
Weighted Avg	0.58	0.61	0.59	151
AUC			0.53	151

K =3 threshold = 0.33	Precision	Recall	F1-score	Support
Class 1	0.76	0.60	0.67	100
Class 2	0.44	0.63	0.52	51
Accuracy			0.61	151
Macro Avg	0.60	0.61	0.60	151
Weighted Avg	0.65	0.61	0.62	151
AUC			0.63	151

K =5 threshold = 0.3	Precision	Recall	F1-score	Support
Class 1	0.74	0.74	0.74	100
Class 2	0.49	0.49	0.49	51
Accuracy			0.66	151
Macro Avg	0.62	0.62	0.62	151
Weighted Avg	0.65	0.66	0.66	151
AUC			0.64	151

K =10 threshold = 0.3	Precision	Recall	F1-score	Support
Class 1	0.76	0.83	0.79	100
Class 2	0.60	0.49	0.54	51
Accuracy			0.72	151
Macro Avg	0.68	0.66	0.67	151
Weighted Avg	0.71	0.72	0.71	151
AUC			0.70	151

K =15 threshold = 0.33	Precision	Recall	F1-score	Support
Class 1	0.79	0.83	0.81	100
Class 2	0.63	0.57	0.60	51
Accuracy			0.74	151
Macro Avg	0.71	0.70	0.70	151
Weighted Avg	0.74	0.74	0.74	151
AUC			0.72	151

K =17 threshold = 0.33	Precision	Recall	F1-score	Support
Class 1	0.78	0.88	0.83	100
Class 2	0.68	0.51	0.58	51
Accuracy			0.75	151
Macro Avg	0.73	0.69	0.71	151
Weighted Avg	0.75	0.75	0.74	151
AUC			0.72	151

K =20 threshold = 0.33	Precision	Recall	F1-score	Support
Class 1	0.77	0.89	0.82	100
Class 2	0.69	0.47	0.56	51
Accuracy			0.75	151
Macro Avg	0.73	0.68	0.69	151
Weighted Avg	0.74	0.75	0.73	151
AUC			0.71	151

It is not hard to see that as K gets closer and closer to 16, the overall performance of the model gets better and better, which is consistent with our expectations.

In addition, it is worth noting that the better threshold value is basically 0.33, which may be because our data is very unbalanced, that is to say:

$$\frac{\#Class1}{\#Class2} \approx \frac{2}{1}$$

So it is likely that the ratio near the boundary point still be close to 2:1, so threshold is a good choice.

5.2 Least Square Model

5.2.1 Least square without regularization

The method of **least squares** is a standard approach in regression analysis to approximate the solution of overdetermined systems (sets of equations in which there are more equations than unknowns) by minimizing the sum of the squares of the residuals.

$$residual_i = y_i - f(x_i, \beta)$$
$$S = \sum_i (residual_i^2) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$$
$$\beta_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Here, the dimantion of y_i is (1,2) and the meaning of it is the score for getting 0 and the score for getting 1.

```
def trainning_least_square(trainning_set):
    y_vec = trainning_set['class']
    x_matrix = trainning_set.drop(['class'], axis = 1)
    # you can use this to standardize the data if no preprocessing
    # x_matrix = x_matrix.apply(lambda t:(t - np.min(t)) / (np.max(t)
- np.min(t)))
    # x_matrix = x_matrix.apply(lambda t:((t - np.mean(t)) /
np.std(t)))
    x_matrix['one'] = 1
    y_vec = classOneHot(y_vec)
    x_matrix = np.array(x_matrix)
    x_matrix = x_matrix.T
    first_part = np.dot(y_vec, x_matrix.T)
    second_part = np.linalg.inv(np.dot(x_matrix, x_matrix.T))
    beta_hat = np.dot(first_part, second_part)
    return beta_hat
```

The result is:

AUC	F1-score	Precision	Recall	Accuracy
0.7967	0.82	0.71	0.98	0.72
	0.34	0.85	0.22	

5.2.2 Least square with L2 regularization

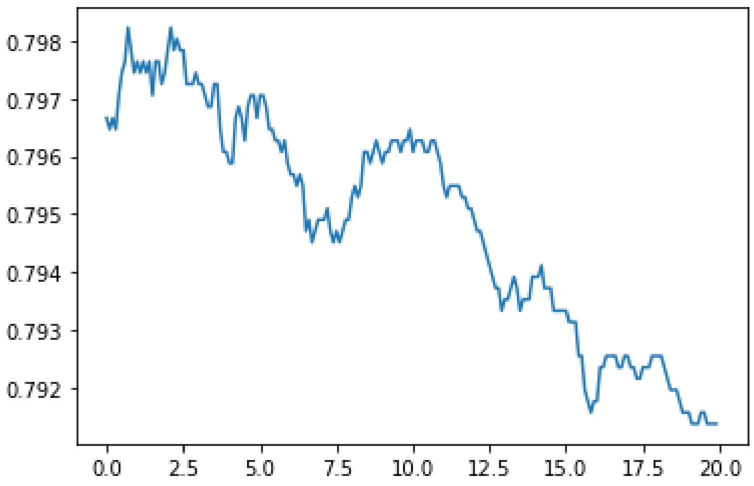
The formula of least square with L2 regularization is:

$$S(\beta) = \sum_i residual_i^2 + \lambda \sum_i \beta_i^2 = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta^T\beta$$

$$\beta_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} + \lambda\mathbf{I}\beta$$

```
    y_vec = trainning_set['class']
    x_matrix = trainning_set.drop(['class'], axis = 1)
    # x_matrix = x_matrix.apply(lambda t:((t - np.mean(t)) /
np.std(t)))
    x_matrix['one'] = 1
    y_vec = classOneHot(y_vec)
    x_matrix = np.array(x_matrix)
    x_matrix = x_matrix.T
    beta_hat = np.dot(np.dot(y_vec, x_matrix.T),
                      np.linalg.inv(np.dot(x_matrix, x_matrix.T)+
                      lam * np.diag(np.ones(len(trainning_set.T)))))
```

Using the AUC as the criteria of choosing λ, we get the graph:



The results shows that when $\lambda = 2.1$, it gets the best AUC:

AUC	F1-score	Precision	Recall	Accuracy
0.7982	0.82	0.71	0.98	0.72
	0.32	0.83	0.20	

It says that adding regularization does no help to the model fitting. The possible reasons are: (1) we use the dataset after PCA, so adding L2 regularization does function on the features selection; (2) the data has a small overlap between the two categories, which is difficult to distinguish by linearity and simple nonlinearity.

5.2.3 The predictions on the training set: without and with $\lambda = 2.1$

The model performance on training data is:

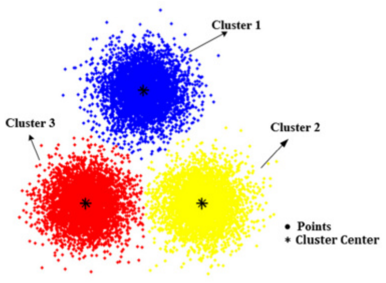
λ	AUC	F1-score	Precision	Recall	Accuracy
0	0.7680	0.88	0.80	0.96	0.79
		0.33	0.64	0.22	
2.1	0.7688	0.88	0.80	0.96	0.79
		0.32	0.63	0.22	

5.3 LS with Gaussian Basis Model

5.3.1 How to Choose Appropriate μ and σ^2

5.3.1.1 K-means Clustering

This is an appropriate way to determine μ of the Gaussian Basis functions. It will cluster data points with similar features into the same group, then I can set the values of centroid data in each group as μ for the Gaussian Basis functions. The data I use can represent the average performance of one group of data.



I will only use 64 attributes from data to do K-means clustering. Here, I will divide the data into 20 groups, and use the data of the centroid of each group to form the Gaussian function. Here, I will use `KMeans` function from `sklearn.cluster` to do the clustering. The code is the following:

```
def kClusteringMean(self, n):
    """
    Use k-clustering method to find 20 mean values
    """
    train_adj = self.train_x.copy()
    train_adj["cluster"] = KMeans(n_clusters=n, max_iter=10000,
random_state=119020).fit_predict(self.train_x)
    train_adj_mean = train_adj.groupby("cluster").mean()
    self.k_cluster_mean = train_adj_mean
```

5.3.1.2 Random Generate

First, I will calculate mean and variance of each attributes, and use these data to form 64 normal distribution functions. After that, I will randomly generate 80 points from each normal distribution functions. Therefore, I get a `80×64` mean value matrix. Here, I will use `numpy.random.normal` function to randomly generate data from the given normal distribution. The code I use is as the following:

```
def normalRandomMean(self, n, seed):
    mean_var_lst = list(zip(self.train_x.mean(),
np.sqrt(self.train_x.var())))
    zeros = np.zeros([n, len(mean_var_lst)])
    for attr in range(len(mean_var_lst)):
        np.random.seed(seed)
        zeros[:,attr] = np.random.normal(mean_var_lst[attr][0],
mean_var_lst[attr][1],n)
    zeros = pd.DataFrame(zeros, columns = self.k_cluster_mean.columns)
    self.mean_generate = pd.concat([pd.DataFrame(zeros),
self.k_cluster_mean])
```

5.3.1.3 Choose an appropriate σ^2

Here, I use the variance of the origin data as the σ^2 I use in the Gaussian basis functions.

5.3.2 Using Gaussian Basis Functions to Transform Data

From *2.1.1*, I have generated a `100×64` mean value matrix, and a constant variance. I will use the following formula to transform the origin data:

$$\phi_j(x) = exp\{-\frac{(x-\mu_j)^2}{2\sigma^2}\}$$

After that, I will get a new training set with the size of `100×949×64`. The code I use is:

```
def gaussianTransform(self, row):
    gaussian_x_train = np.exp(- (self.train_x - 
self.mean_generate.iloc[row,])**2 / 2 * self.train_x.var())
    guassian_x_test = np.exp(- (self.test_x - 
self.mean_generate.iloc[row,])**2 / 2 * self.test_x.var())
    gaussian_x_train["constant"] = 1
    guassian_x_test["constant"] = 1
    self.gaussian_x_train = np.array(gaussian_x_train).T
    self.gaussian_x_test = np.array(guassian_x_test).T
```

5.3.3 Calculate Parameters

After getting the transformed data from Gaussian basis functions, I can use similar method in Least Square method to calculate the data. After using one-hot method to transform the origin test data y(with size `949×1`) into new test data T (with size `949×2`). Then I use the following formula to calculate the parameters:

$$w_{LS} = (X^TX)^{-1}X^TT$$

The code is the following:

```
def gaussianTrain(self, row):
    self.gaussianTransform(row)
    T_XT = np.dot(self.train_y, self.gaussian_x_train.T)
    X_XT_inv = np.linalg.pinv(np.dot(self.gaussian_x_train, 
self.gaussian_x_train.T))
    self.parameters = np.dot(T_XT, X_XT_inv)
    self.gaussian_predict()
```

5.3.4 Predict Result

From *2.1.3*, I get the parameter of the least square method, then I can use the formula to do the prediction:

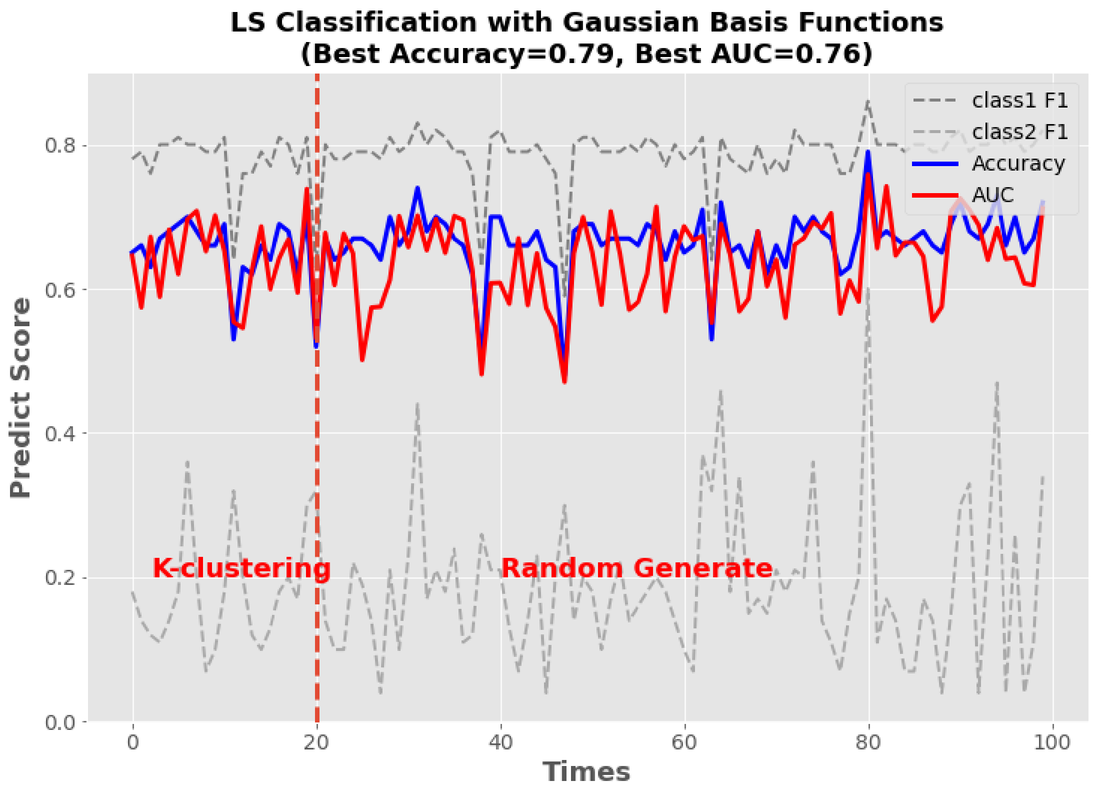
$$\hat{y}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$$

Now, I will get a `2×151` prediction method, with each column is the predict value of the given test data. For each column $\mathbf{t} = [t_1, t_2]^T$, I will compare the predict values under two parameters. I will predict the data as `0` if $t_1 > t_2$, and predict it as `+1` otherwise. After that, I will calculate `F1-score`, `Accuracy` and `AUC` of the prediction. The code I use is:

```
def gaussian_predict(self):
    df = pd.DataFrame(np.dot(self.parameters, self.gaussian_x_test)).T
    df["predict"] = 0
    df.loc[df[1]>df[0], "predict"] = 1
    target_names = ['class 1', 'class 2']
    result = classification_report(np.array(self.test_y),
df['predict'], target_names=target_names)
    self.auc_score = roc_auc_score(np.array(self.test_y),
df['predict'])
    self.result = result
    self.recordDict["class1 F1"].append(float(self.result.split("\n")
[2].split()[4]))
    self.recordDict["class2 F1"].append(float(self.result.split("\n")
[3].split()[4]))
    self.recordDict["Accuracy"].append(float(self.result.split("\n")
[5].split()[1]))
    self.recordDict["AUC"].append(self.auc_score)
```

5.3.5 Result Analysis

In total, I tried 100 different Gaussian Basis Functions, with 100 different μ and 1 constant σ^2. Among all the 100 models, the average accuracy is **66.45%**, the average AUC (Area Under the Curve) is **63.85%**. The result is as the following:



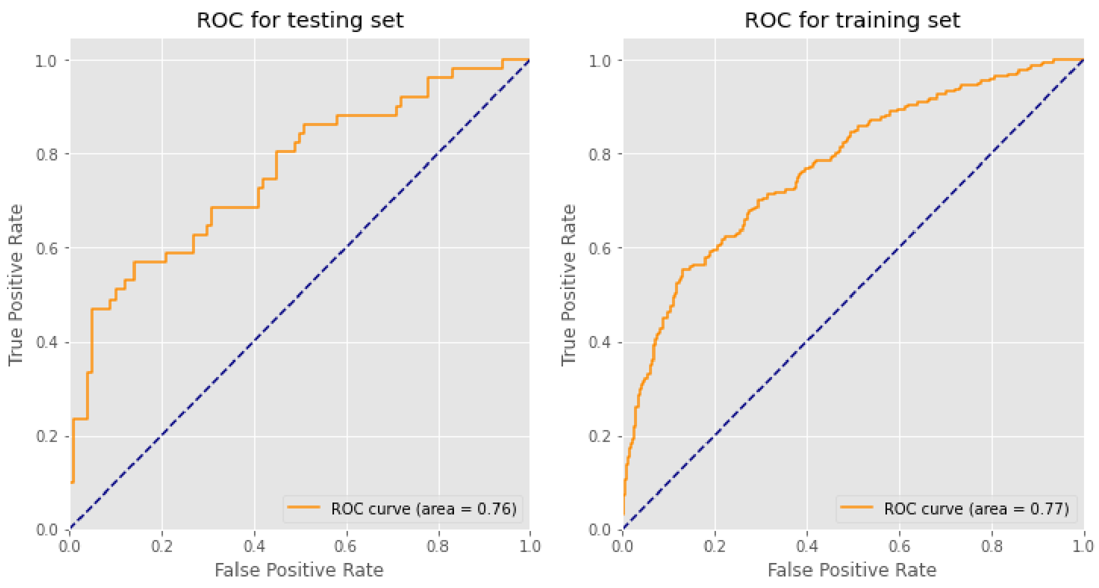
This average prediction results are relatively lower than other models because:

1. In `k-clustering method`, some groups only contain several points, which makes the mean value can not represent the whole data set. Therefore, the Gaussian transformation will make the prediction worse.
2. In `randomly generating method`, sometimes it will generate illogical data, therefore using this data to do the Gaussian transformation will make the prediction worse.

However, the best prediction model using LS with Gaussian basis function can have **79.23%** accuracy in testing, with **76.37%** AUC, which is pretty good in classification. The result matrix is:

	Precision	Recall	F1-score	Support
Class 1	0.78	0.95	0.86	100
Class 2	0.83	0.47	0.60	51
Accuracy			0.79	151
Macro Avg	0.80	0.71	0.73	151
Weighted Avg	0.80	0.79	0.77	151
AUC			0.76	151

The ROC and AUC of LS with Gaussian basis for testing data and training data are as the following:



5.4 Linear Discriminant Analysis (LDA)

5.4.1 Overview of LDA

Linear Discriminant Analysis (LDA) aims to project data into a dimension that maximizes variance among the different classes and minimizes variance within each of the classes. LDA believes that in this new dimension, better classification can be carried out.

We consider the case of a two-class classification problem, we need to find $\mathbf{w}$ to project data. So, we

$$\max_{w} J(w) = \frac{w^T \mathbf{S}_\mathrm{B} w}{w^T \mathbf{S}_\mathrm{W} w} (Fishercriterion)$$

subject to

$$\begin{aligned}
&\mathbf{m}_k = \frac{1}{N_k}\sum_{n=1}^{N_k} \mathbf{x}_n \quad \text{center of class } \mathbf{k} \\
&\mathbf{S}_\mathrm{B} = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T \text{covariance between class} \\
&\mathbf{S}_\mathrm{W} = \sum_{n \in \mathcal{C}_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^\mathrm{T} + \sum_{n \in \mathcal{C}_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^\mathrm{T} \text{covariance within class}
\end{aligned}$$

This optimization problem can be transformed into a generalized eigenvalue problem.

$$\mathbf{S}_\mathrm{B}\mathbf{w} = J(\mathbf{w})\mathbf{S}_\mathbf{w}\mathbf{w}$$

So w is the eigenvector corresponding to the maximum eigenvalue of $\mathbf{S}_\mathrm{W}^{-1}\mathbf{S}_\mathrm{B}$

The whole process of establishing the LDA of two categories is:

- **Step-1**: Calculate $\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{S}_\mathrm{W}$ and $\mathbf{S}_\mathrm{B}$ by using training data
- **Step-2**: Calculate the eigenvector corresponding to the maximum eigenvalue of $\mathbf{S}_\mathrm{W}^{-1}\mathbf{S}_\mathrm{B}$ as w
- **Step-3**: Find threshold $w_0 = -\frac{w^T\mathbf{m}_1 + w^T\mathbf{m}_2}{2}$
- **Step-4**: For new data x, if $\frac{w_0 + w^T x}{w_0 + w^T \mathbf{m}_2} > 0$, we put it in class 2. Otherwise, we put it in class 1

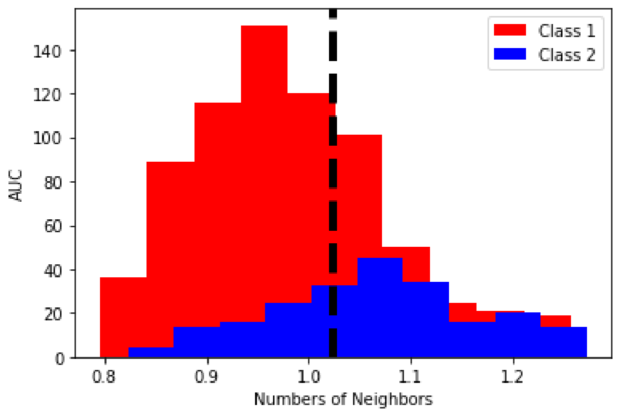
See appendix for code.

5.4.2 Result Analysis

Now, we use LDA in our pre-processed data. In the training dataset we got the following results:

Training Data	Precision	Recall	F1-score	Support
Class 1	0.87	0.69	0.77	100
Class 2	0.39	0.67	0.50	51
Accuracy			0.68	151
Macro Avg	0.63	0.68	0.63	151
Weighted Avg	0.76	0.68	0.71	151
AUC			0.73	151

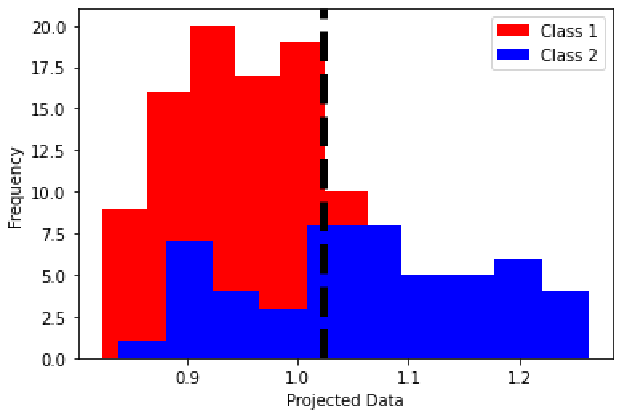
The figure below shows the projected data and decision boundaries in the training dataset.



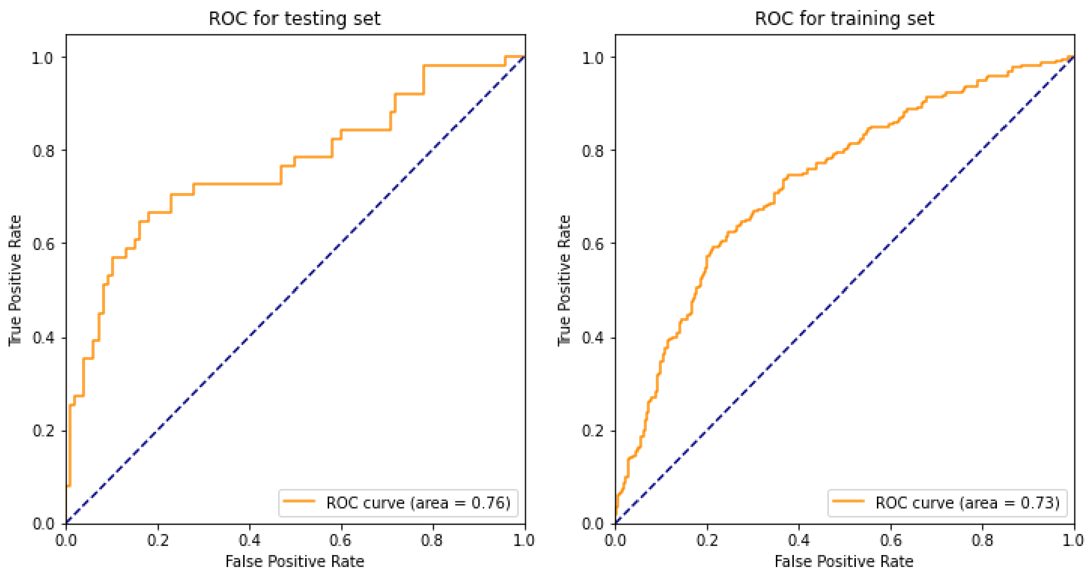
In the testing dataset we got the following results:

Testing Data	Precision	Recall	F1-score	Support
Class 1	0.82	0.80	0.81	100
Class 2	0.63	0.67	0.56	51
Accuracy			0.75	151
Macro Avg	0.73	0.73	0.73	151
Weighted Avg	0.76	0.75	0.76	151
AUC			0.76	151

The figure below shows the projected data and decision boundaries in the testing dataset.



Here is the ROC curve:



We were surprised to find that the models on the test dataset performed even better than those on the training dataset. In the projected distribution, we can find that the test dataset has a distribution that is more consistent with our expectations, but in the training dataset, there are a lot of confounding data near the decision boundary, which leads to better performance of our model on the test dataset. This is a coincidence caused by the inconsistent distribution of training and test data sets.

But overall, the results of the test set are satisfactory. It has a good recall rate(67%) for class 2 , and a high AUC(76%) and accuracy rate(75%) overall.

5.5 Logistic classification with/without regularization (L2).

5.5.1 Logistic classification without regularization

The binary logistic regression model in the general form:

$$y = f(\mathbf{x}; \beta) + \varepsilon$$

The output y takes on two possible outcomes: "is the case (1)" and "not the case (2)". Then we assume that the output y given a specific x is a Bernoulli random variable with the probability mass function as follows:

y	Probability mass function
1	$P(y = 1\|\mathbf{x}) = p$
0	$P(y = 0\|\mathbf{x}) = 1 - p$

Due to the Bernoulli model assumption, we have:

$$\begin{aligned} y &= f(\mathbf{x}; \beta) + \varepsilon \\ &= E(y|\mathbf{x}, \beta) + \varepsilon \\ &= P(y = 1|\mathbf{x}) + \varepsilon \end{aligned}$$

$P(y = 1|x) = p$ can be regarded as "the probability that the output y takes outcome given the inputs x is equal to p". It is most widely used that:

$$P(y = 1|x) = e^{\mathbf{x}^T\beta}/(1 + e^{\mathbf{x}^T\beta}) = \frac{1}{1 + e^{\mathbf{x}^T\beta}}$$

Then we fit model using MLE Estimation: given the set of 996 data points in $S = (y_1, x_1), (y_2, x_2), (y_3, x_3) \ldots (y_9 96, x_9 96)$, the likelihood function is given:

$$L(y|X, \beta) = L(y_1, y_2 \ldots y_{949}|X, \beta) = \prod_i P(y_i|x_i) = \prod_i p_i^{y_i}(1 - p_i)^{1-y_i}$$

$$I(\beta) = logL = \sum_i ln(1 - p_i) + \sum_i y_i ln(\frac{p_i}{1 - p_i})$$

$$= \sum_i ln(1 - \frac{1}{1 + e^{\mathbf{x}^T\beta}}) + \sum_i y_i ln(\frac{1}{1 + e^{\mathbf{x}^T\beta}}/(1 - \frac{1}{1 + e^{\mathbf{x}^T\beta}})$$

We can clearly see that the partial derivative of $I(\beta)$ does not contain beta, so we choose to use gradient descent method to find the suitable beta which gives the lower loss value:

$$Loss = \sum_i (\hat{y}_i - y_i)^2$$

```
# Loss function
def square_loss(y_pred, target):
  return np.mean(pow((y_pred - target),2))
# Gradient Descent
for i in range(10000):
    gradient_W = np.dot((y_pred-y_tr).T, X_tr)/(X_tr.shape[0])
    gradient_b = np.mean(y_pred-y_tr)
    W = W - lr * gradient_W
    b = b - lr * gradient_b
    z = np.dot(X_tr, W) + b
    y_pred = sigmoid(z)
    loss.append(square_loss(y_pred, y_tr))
```

The prediction result shows:

AUC	F1-score	Precision	Recall	Accuracy
0.7945	0.84	0.72	0.99	0.75
	0.40	0.93	0.25	

5.5.2 Logistic classification with regularization

The constraint represented by the l2 regularization is a hypersphere in high-dimensional space, and since there are no corners, the probability of the first intersection occurring at a location with sparsity becomes very small, tending to obtain smaller numbers that are all non-zero and therefore smooth.

In addition to preventing overfitting, l2 regularization can also be used as a feature filtering method, so that the weight coefficients of features that are not too important to the model tend to zero, and then we can remove and reselect features according to the specific situation, thus playing a role in improving generalization performance as well as saving memory space and improving operational efficiency.

The log likelihood function becomes:

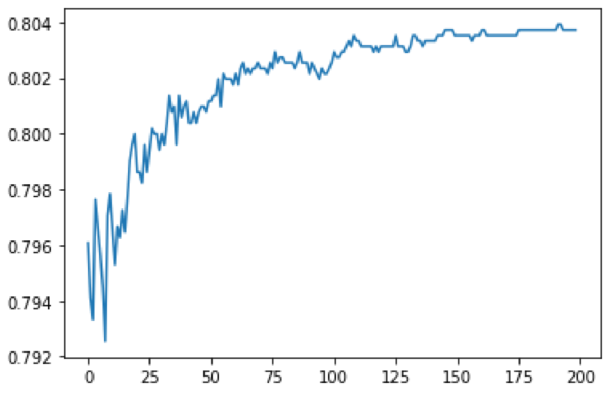
$$I(\beta) = logL = \sum(ln(1-p_i)) + \sum y_i ln(\frac{p_i}{1-p_i}) + \lambda \sum \beta_j$$

$$\frac{\partial I(\beta)}{\partial \beta} = (\mathbf{p}^T - \mathbf{y}^T)\mathbf{X} + \lambda\beta$$

Now, we still need to use gradient descent method to find the best \beta. And in the gradient descent formula, we need to add the L2 norm term :

```
for i in range(10000):
        gradient_W = (np.dot((y_pred-y_tr).T, X_tr) + lam *
W)/(X_tr.shape[0])
        gradient_b = np.mean(y_pred-y_tr)
        W = W - lr * gradient_W
        b = b - lr* gradient_b
        z = np.dot(X_tr, W) + b
        y_pred = sigmoid(z)
        loss.append(square_loss(y_pred, y_tr))
```

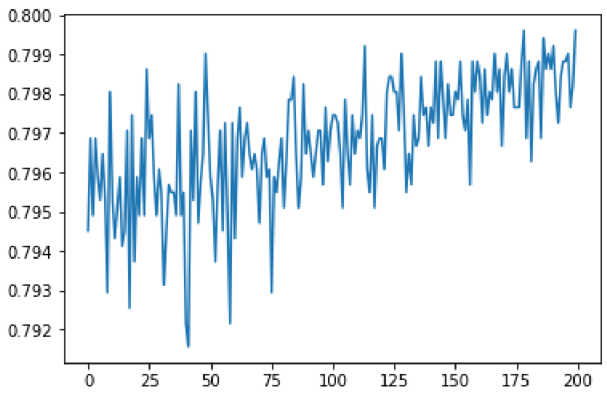
We use for loop to search for the best λ from 1 to 200 with step 1 and according to the graph:



Then we check the f1 score, precision, recall and accuracy, there exists a trade off between AUC and them:

λ	AUC	F1-score	precision	Recall	Accuracy
1	0.796	0.83	0.72	0.99	0.74
		0.38	0.92	0.24	
30	0.800	0.83	0.7	1.00	0.72
		0.3	1.00	0.18	
100	0.8025	0.81	0.68	1.00	0.68
		0.11	1.00	0.06	
170	0.8035	0.8	0.67	1.00	0.67
		0.04	1.00	0.02	

Obviously, adding L2 regularization does no help toward the Logistic classification: while little increasing in the AUC, there is a huge decrease in F1 score. Let's try small pieces of λ from (0, 20) to see the behaviors of AUC:



The graph shows that the existence of λ leads to random movement of the AUC, meaning that the L2 regularization almost does not work. The possible reasons are: (1) we use the dataset after PCA, so adding L2 regularization does function on the features selection; (2) the data has a small overlap between the two categories, which is difficult to distinguish by linearity and simple nonlinearity. Then we just use the Logistic regression model without regularization. The prediction outcomes in the training set is:

AUC	F1-score	Precision	Recall	Accuracy
0.7720	0.88	0.81	0.95	0.80
	0.38	0.64	0.27	

5.6 SoftMax Classification Model

5.6.1 Overview of SoftMax Model

SoftMax model is the generalized form of logistic classification model. It can do the multi-class classification, while the logistic model can only be applied to binary classification problem. In SoftMax model, for the given data $\{x_1, y_1), \ldots, (x_n, y_n)\}$ with k classes, we should first calculate the probability $P(x_i) \in class_k|\theta$ for the given parameter $\theta = [\theta_1, \ldots, \theta_n]$ using the following formula:

$$h_\theta\left(x_i\right) = \begin{bmatrix} p\left(y_i = 1|x_i;\theta\right) \\ p\left(y_i = 2|x_i;\theta\right) \\ \vdots \\ p\left(y_i = k|x_i;\theta\right) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x_i}} \begin{bmatrix} e^{\theta_1^T x_i} \\ e^{\theta_2^T x_i} \\ \vdots \\ e^{\theta_k^T x_i} \end{bmatrix} \tag{1}$$

After that, we want to maximize the likelihood function for the training data, given parameters θ:

$$maximize \prod_{i=1}^{n} h_\theta(x_i)$$

I will use the Gradient Descent method to find the parameters.

5.6.2 Use Gradient Descent to Find Parameters:

I will define the negative logarithm of likelihood (cross-entropy error function) as the loss function. Here, the loss function is:

$$L(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\left\{y_i = j\right\} \log \frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k} e^{\theta_l^T x_i}}\right] \tag{4}$$

By taking gradient for each θ_j, I get the gradient using the following formula[1]:

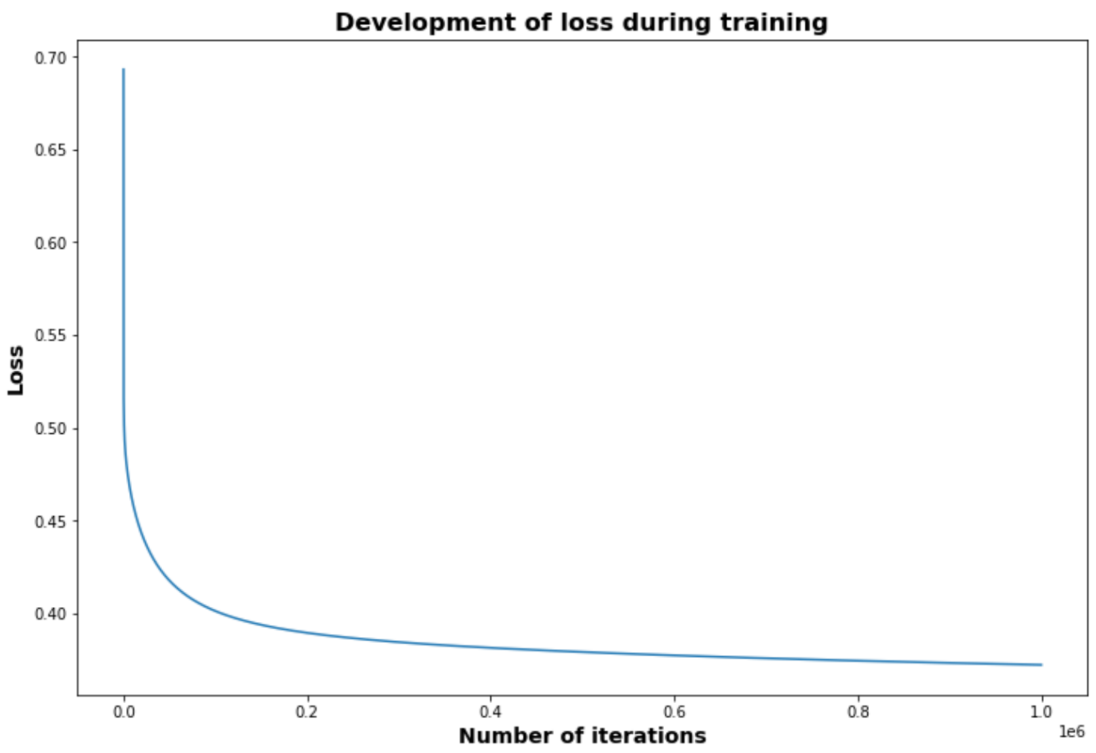
$$
\begin{aligned}
\frac{\partial L(\theta)}{\partial \theta_j} &= -\frac{1}{m}\frac{\partial}{\partial \theta_j}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\left\{y_i = j\right\}\log\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k} e^{\theta_l^T x_i}}\right] \\
&= -\frac{1}{m}\frac{\partial}{\partial \theta_j}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\left\{y_i = j\right\}\left(\theta_j^T x_i - \log\sum_{l=1}^{k} e^{\theta_l^T x_i}\right)\right] \\
&= -\frac{1}{m}\left[\sum_{i=1}^{m} 1\left\{y_i = j\right\}\left(x_i - \sum_{j=1}^{k}\frac{e^{\theta_j^T x_i}\cdot x_i}{\sum_{l=1}^{k} e^{\theta_l^T x_i}}\right)\right] \\
&= -\frac{1}{m}\left[\sum_{i=1}^{m} x_i 1\left\{y_i = j\right\}\left(1 - \sum_{j=1}^{k}\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k} e^{\theta_l^T x_i}}\right)\right] \qquad (5) \\
&= -\frac{1}{m}\left[\sum_{i=1}^{m} x_i\left(1\left\{y_i = j\right\} - \sum_{j=1}^{k} 1\left\{y_i = j\right\}\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k} e^{\theta_l^T x_i}}\right)\right] \\
&= -\frac{1}{m}\left[\sum_{i=1}^{m} x_i\left(1\left\{y_i = j\right\} - \frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k} e^{\theta_l^T x_i}}\right)\right] \\
&= -\frac{1}{m}\left[\sum_{i=1}^{m} x_i\left(1\left\{y_i = j\right\} - p\left(y_i = j | x_i; \theta\right)\right)\right]
\end{aligned}
$$

I will get the parameters using the following steps:

1. Randomly choose the initial parameters θ_0.
2. Calculate the loss function $L(\theta)$ mentioned above.
3. Calculate the gradient descent $\Delta\theta$ using the above formula.
4. If $\Delta\theta < \varepsilon$, then stop and return the parameter. Otherwise, continue.
5. Update parameter θ using $\theta_{n+1} = \theta_n - \alpha\Delta\theta_n$.
6. Update softmax values using the new parameter and the formula (1) mentioned above.
7. Repeat step 2~6 until it reaches the maximum iteration times or the loss function converges.

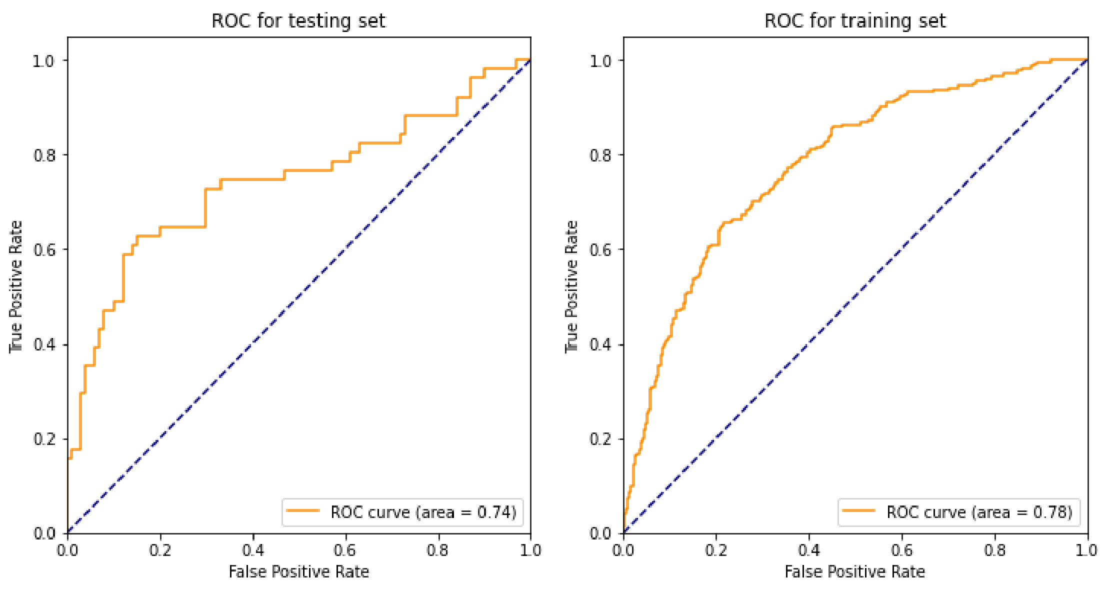
5.6.3 Result Analysis

The gradient descent method will converge after 1,000,000 iterations. The loss function is visualized as:



The accuracy of classification is **75.39%** with AUC **68.15%**. The performance of softmax method is pretty good for the classification problem. The result matrix is:

	Precision	Recall	F1-score	Support
Class 1	0.71	0.96	0.82	100
Class 2	0.75	0.24	0.36	51
Accuracy			0.75	151
Macro Avg	0.74	0.68	0.69	151
Weighted Avg	0.75	0.75	0.74	151
AUC			0.75	151



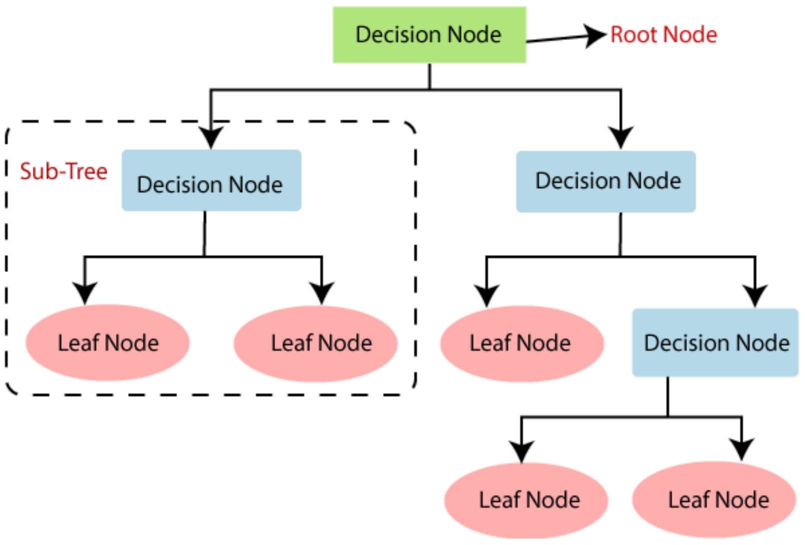
5.6.4 Code of SoftMax method

Please refer to the appendix.

5.7 Decision Tree

5.7.1 Overview of Decision Tree

Decision Tree is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.



The general process of decision tree is:

- **Step-1**: Begin the tree with the root node, says S, which contains the complete training dataset.

- **Step-2**: Split the node according to its attributes and work out the impurity (entropy) for each subset of every attribute.
- **Step-3**: Compute the information gain for each attribute.
- **Step-4**: Generate the decision tree node, which contains the best attribute with maximum information gain.
- **Step-5**: Repeat step-2 to step-4, until a stage is reached where we cannot further classify the nodes and called the final node as a leaf node.

We use sklearn to implement Decision Tree. See appendix for code.

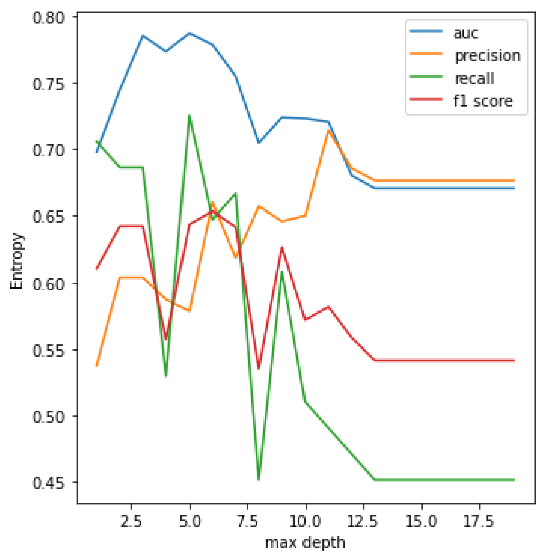
5.7.2 Result Analysis

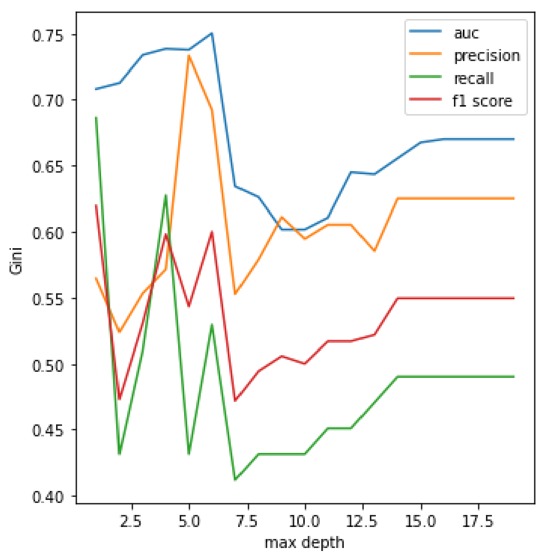
Now, we use Decision Tree in our imputed data since we want to use 100% information.

The decision tree model has many hyperparameters to choose from, and we use the following three important hyperparameters to optimize the model:

- Split Criterion: Split criterion is the parameter used to determine how impurity is calculated. We have two choices: Entropy and Gini
- Max Depth: Limit the maximum depth of the tree, and all branches exceeding the set depth are cut off to prevent over-fitting of the model.
- Min Samples Leaf: Each child node of a node after sorting must contain at least min samples leaf training samples, otherwise splitting will not occur. This is also a way of limiting overfitting

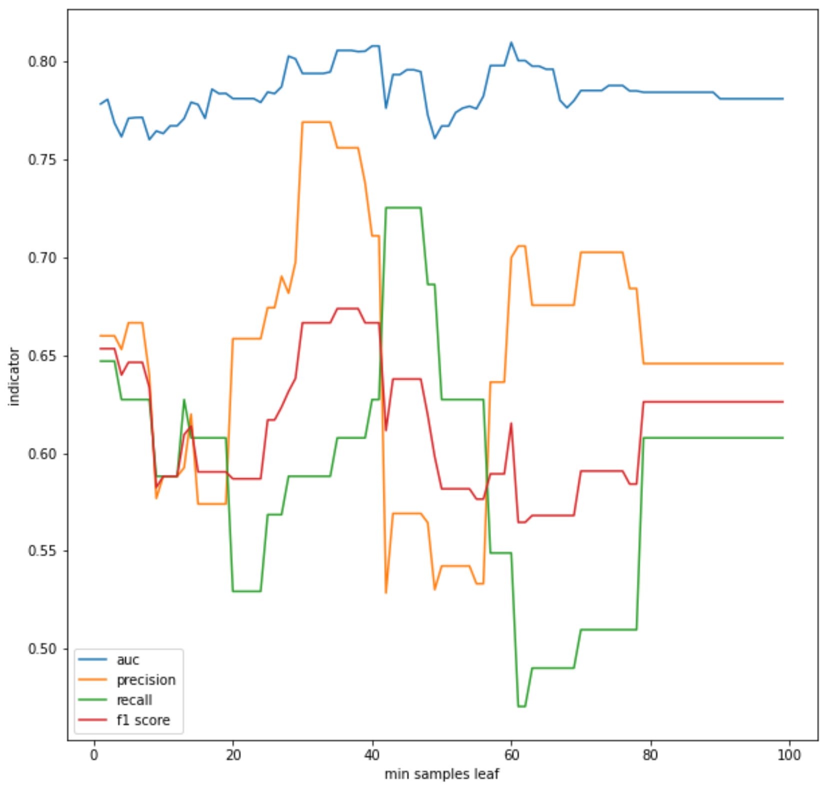
First, we choose Split Criterion and Max Depth. The following are the changes of AUC, precision rate, recall rate and f1-score with Max Depth between 1 to 20 with different Split Criterion in testing data (threshold = 0.3 for the reason which will be explained later):





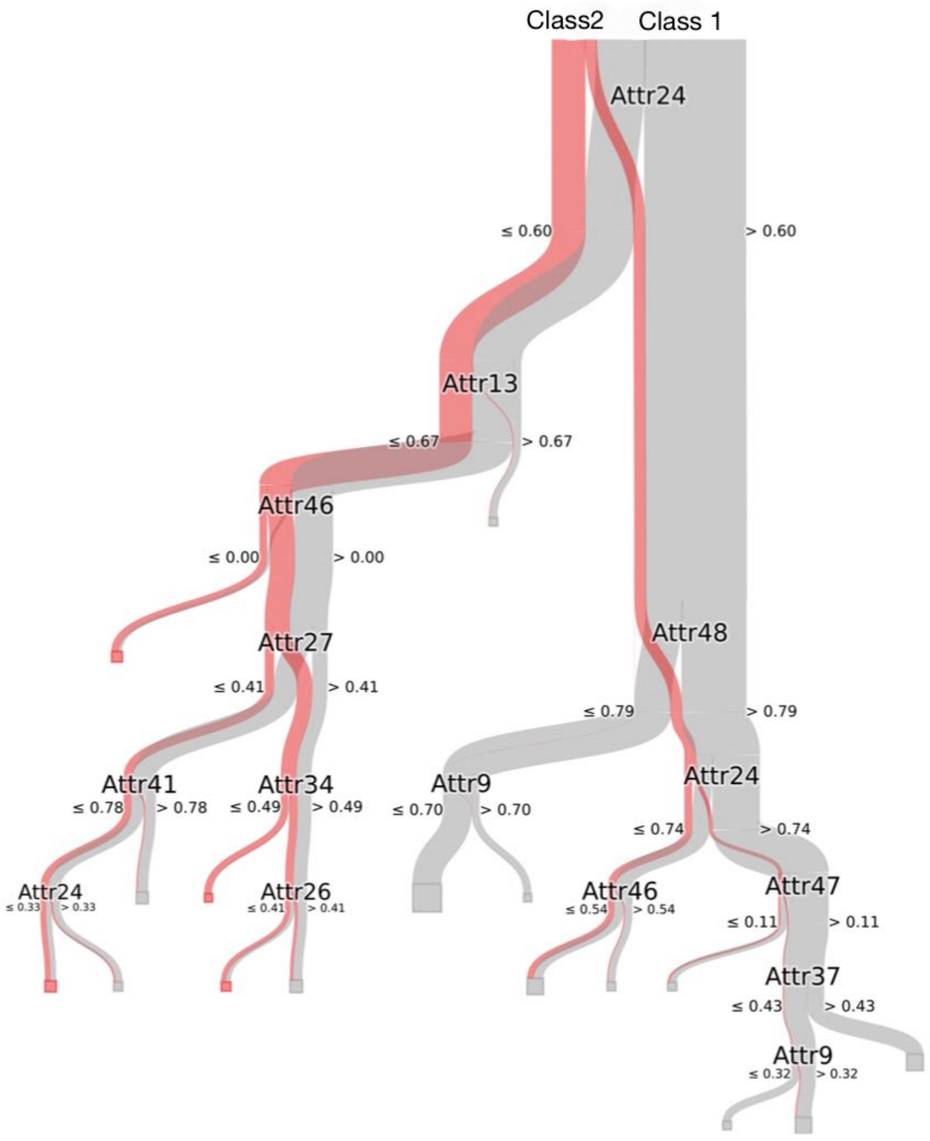
It can be seen that using entropy as the splitting criterion and Max depth = 6 has the highest AUC, and precision rate, recall rate and F1-score are close to the highest.

Then we use entropy as the splitting criterion and Max depth = 6, Min Samples Leaf was changed from 1 to 100 to check the AUC, precision rate, recall rate and F1-score:



It can be seen that using entropy as the splitting criterion, Max depth = 6 and Min sample leaf = 40 has the highest AUC, and precision rate, recall rate and F1-score are close to the highest.

The following is the decision tree model after training with entropy as the splitting criterion, Max depth = 6 and Min sample leaf = 40:



In the training dataset we got the following results:

threshold = 0.5	Precision	Recall	F1-score	Support
Class 1	0.87	0.90	0.89	100
Class 2	0.64	0.57	0.60	51
Accuracy			0.83	151
Macro Avg	0.76	0.74	0.75	151
Weighted Avg	0.82	0.83	0.82	151
AUC			0.87	151

In the testing dataset we got the following results:

threshold = 0.5	Precision	Recall	F1-score	Support
Class 1	0.80	0.94	0.87	100
Class 2	0.82	0.55	0.66	51
Accuracy			0.81	151
Macro Avg	0.81	0.74	0.76	151
Weighted Avg	0.81	0.81	0.80	151
AUC			0.81	151

Since model have a high AUC, we can change the threshold:

In the training dataset we got the following results:

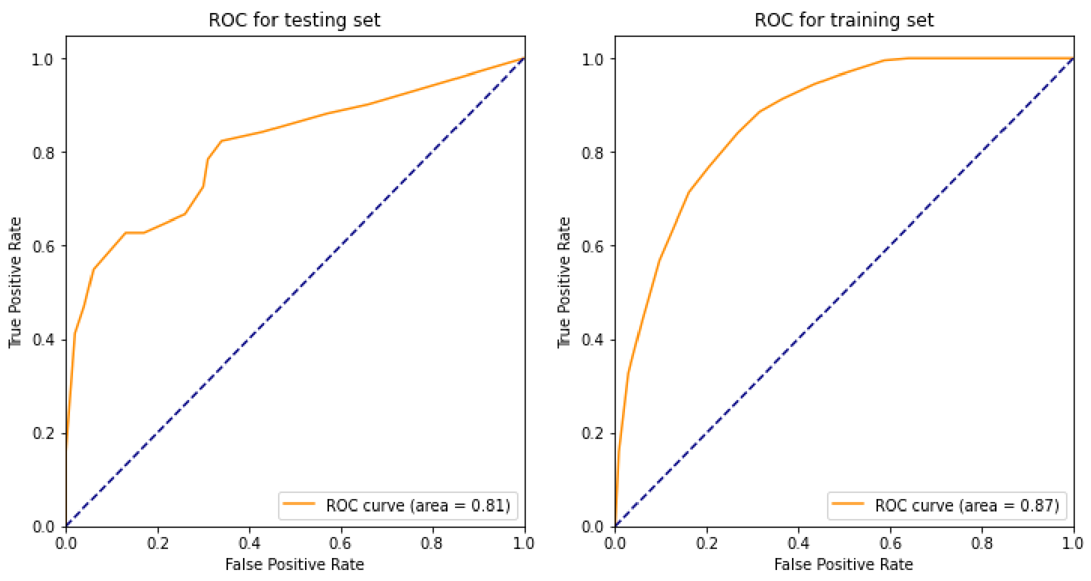
threshold=0.3	Precision	Recall	F1-score	Support
Class 1	0.91	0.84	0.87	100
Class 2	0.57	0.71	0.64	51
Accuracy			0.81	151
Macro Avg	0.74	0.78	0.75	151
Weighted Avg	0.83	0.81	0.82	151
AUC			0.87	151

In the testing dataset we got the following results:

threshold=0.3	Precision	Recall	F1-score	Support
Class 1	0.82	0.87	0.84	100
Class 2	0.71	0.63	0.67	51
Accuracy			0.79	151
Macro Avg	0.77	0.75	0.76	151
Weighted Avg	0.78	0.79	0.78	151
AUC			0.81	151

When the threshold is adjusted to 0.3, we can find that although the overall accuracy is somewhat reduced, the recall rate of Class 2 and F1 score are improved.

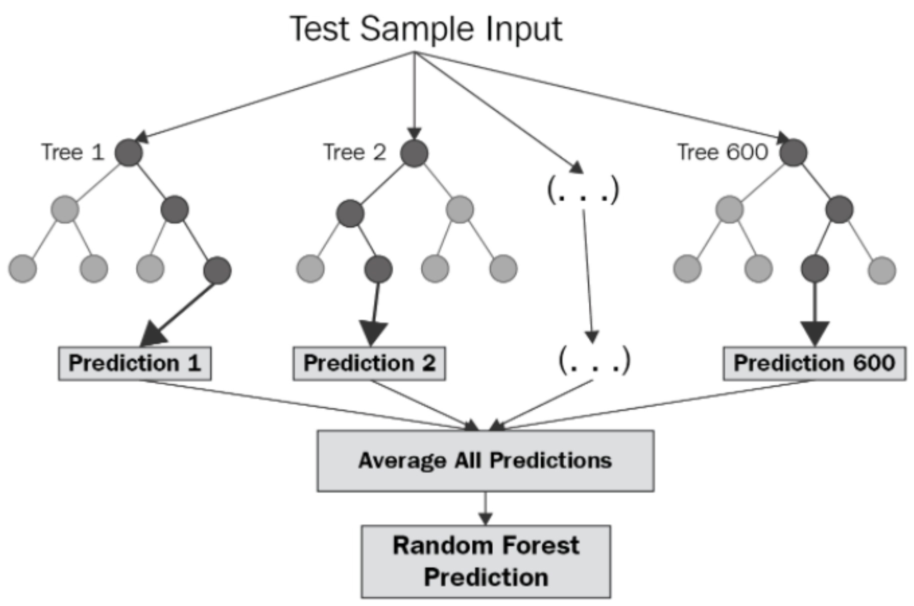
Here is the ROC curve:



5.8 Random Forest Classification Model

5.8.1 Overview of Random Forest Model

Random forest model is randomly creating several decision trees by choosing certain features and samples. After that, using majority voting method to determine the label of the given input. Compared with decision tree model, random forest model is more reliable, stable and robust. However, it may take more time in model training and predicting.

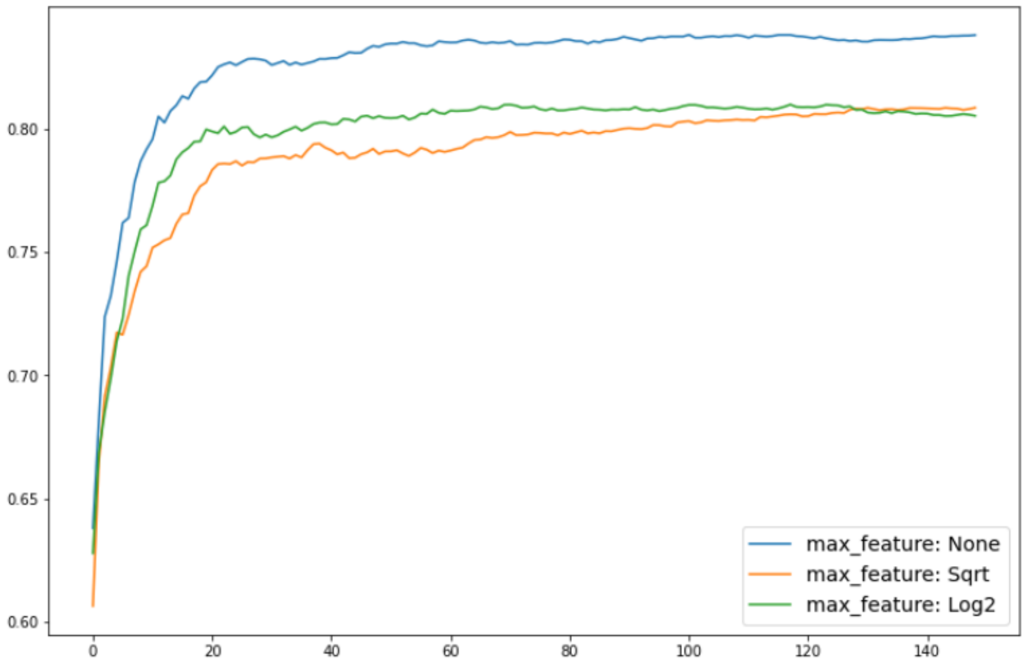


5.8.2 Choose Hyper-parameters

Here, I use `GridSearchCV` method from `sklearn` package to choose hyper-parameters *max_features, criterion, max_feature*. I will use the prediction of testing set to estimate the performance of each hyper-parameter. The evaluation criteria I choose is `AUC`.

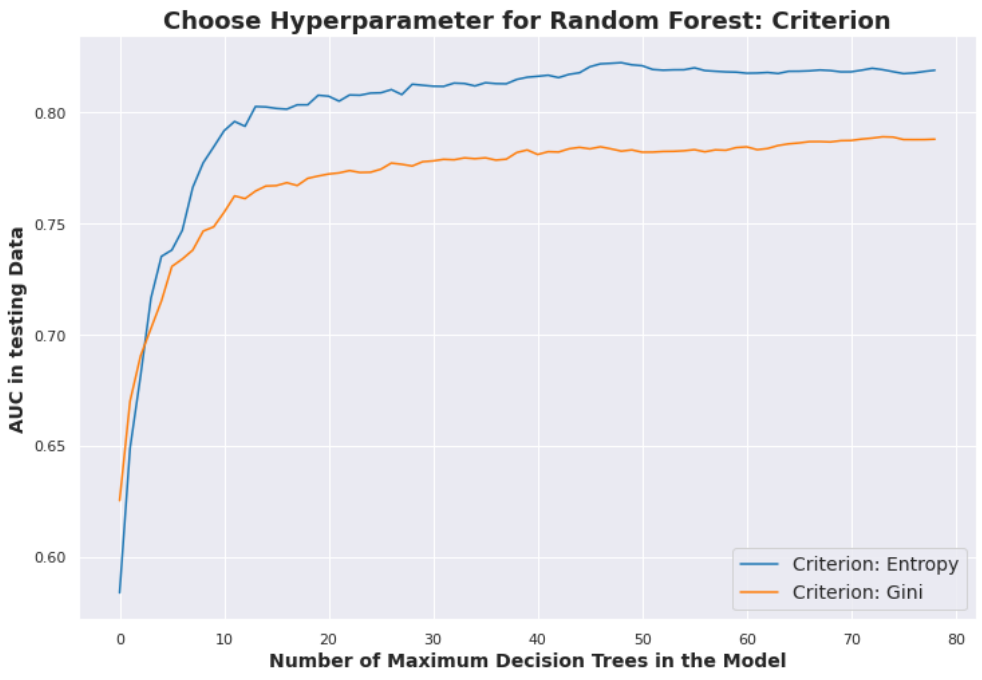
5.8.2.1 Determine maximum feature used in each tree

`max_feature` parameter determines the number of maximum features used in each decision tree of the model. There are 3 choices: not limit, square root of total number of features and logarithm of the total number of features. From the training process I found that we should not limit the maximum features used in the model.



5.8.2.2 Choose Criterion

`Criterion` is the evaluation criteria for the classification random forest to decide how to divide each attribute. I will compare the performance of "Gini Index" and "Entropy", when I change the number of the maximum decision trees in the model.



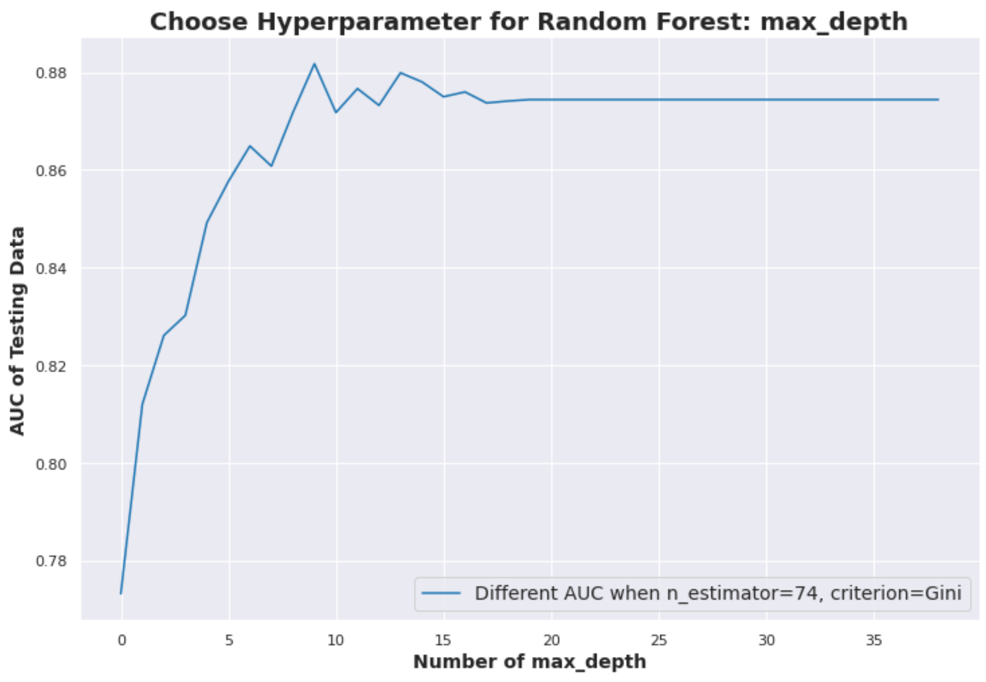
From the above figure I found the performance of "Gini index" is much better than the performance of "Entropy". Thus, I will choose "Gini index" as one of the hyperparameter.

5.8.2.3 Choose number of maximum decision trees in the model

From above two process, when number of maximum decision trees is 74, the model performs the best.

5.8.2.4 Choose the maximum depth of each decision tree

When I set `n_estimator=74, Criterion=Gini, Max_feature=None`, I can view the influence of the maximum depth to the model.



From the above figure I found that when `max_depth=9`, the model performs the best.

5.8.2.5 Choose of threshold

After observing the result of AUC and weighted average F1-score, I think the default threshold 0.5 is not a good estimator for the model to do the prediction. Therefore, I change the threshold from 0.5 to 0.33. And this significantly increase the final F1-score and recall rate.

5.8.2.6 Choose of Hyper-parameters

From the above figure, I can choose the following hyper-parameter to build the `RandomForest` model:

Max_estimator	Criterion	Max_feature	Max_Depth	Threshold
74	Gini	None	9	0.33

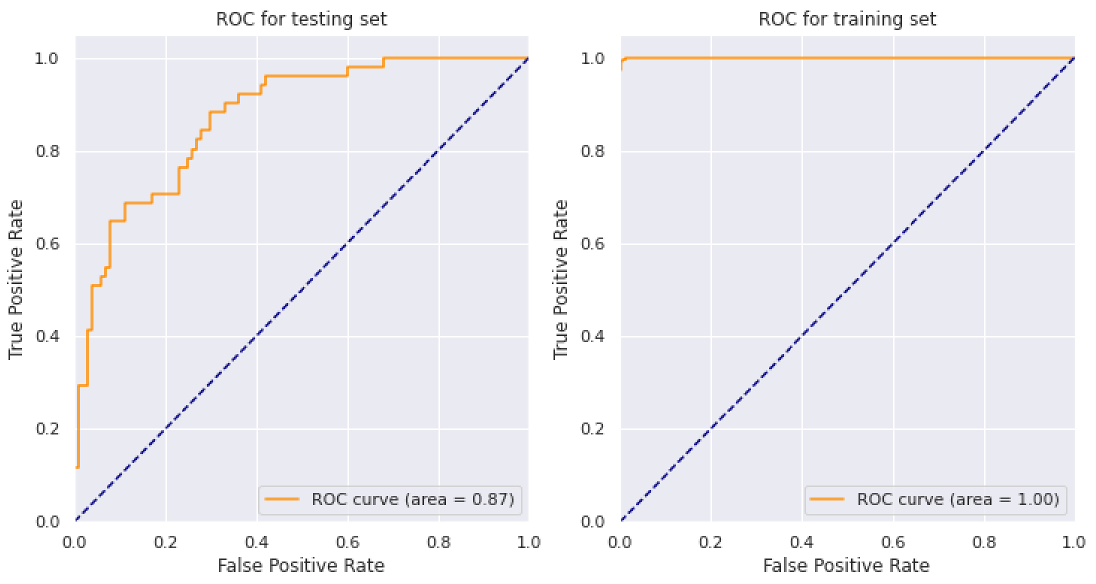
5.8.3 Result Evaluation

Using the above parameters, the model gets 87.17% AUC score in the testing data with 76.12% average accuracy. This is the best model among all the models we get. The performance on testing data is:

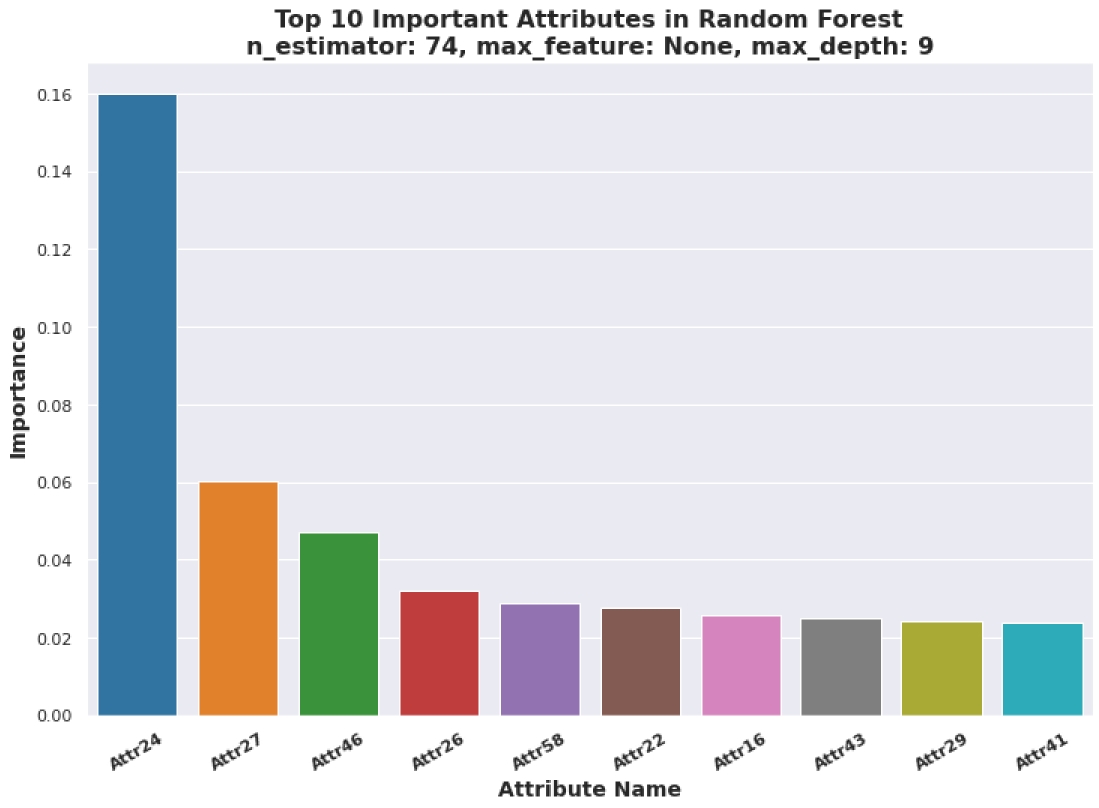
Testing Data	Precision	Recall	F1-score	Support
Class 1	0.75	0.97	0.84	100
Class 2	0.86	0.35	0.50	51
Accuracy			0.76	151
Macro Avg	0.80	0.66	0.67	151
Weighted Avg	0.78	0.76	0.73	151
AUC			0.88	151

The performance on training data is:

Training Data	Precision	Recall	F1-score	Support
Class 1	0.97	1.00	0.98	729
Class 2	1.00	0.88	0.94	220
Accuracy			0.97	949
Macro Avg	0.98	0.94	0.96	949
Weighted Avg	0.97	0.97	0.96	949
AUC			1.00	949



After that, I rank the importance of each attribute in the random forest model. It is as the following:

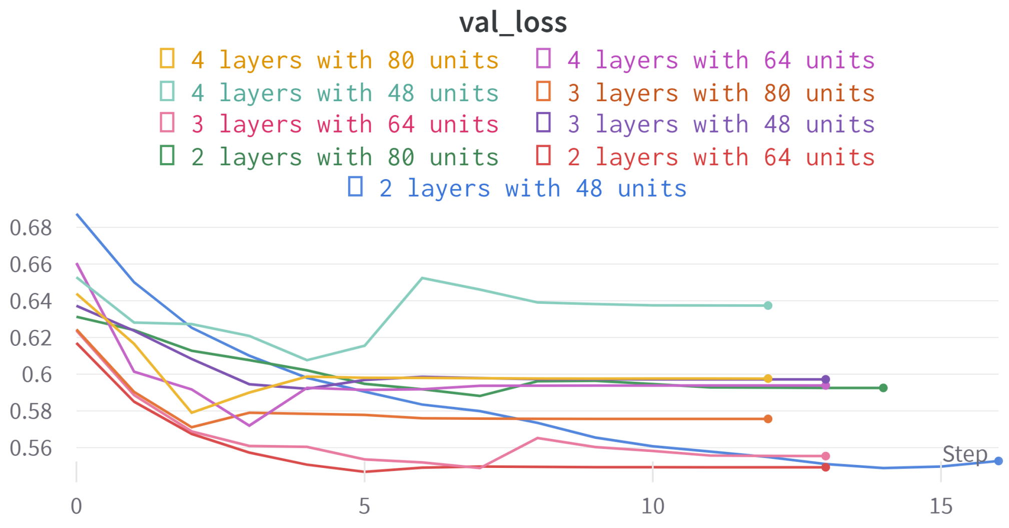


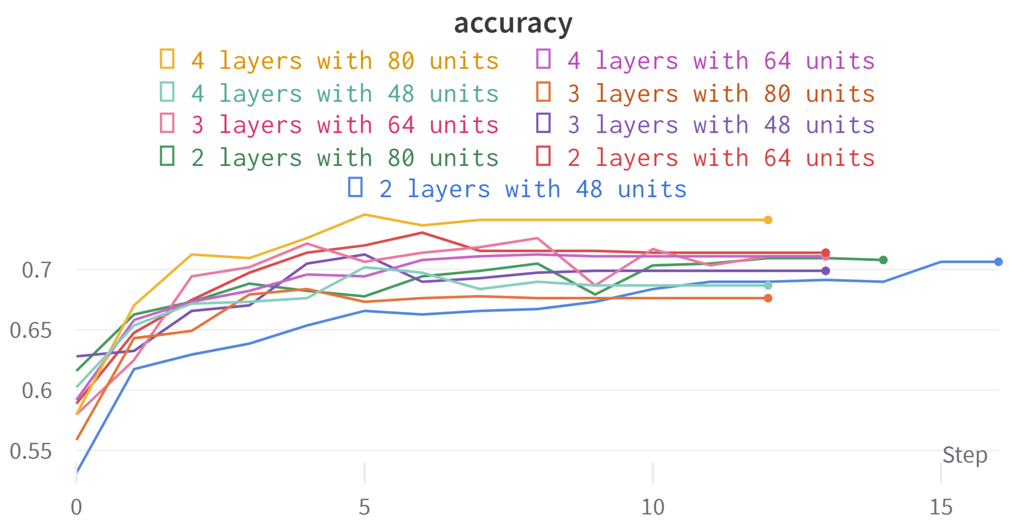
From the above figure, I found that the attribute, which is "Attr24", is of the most important for the model.

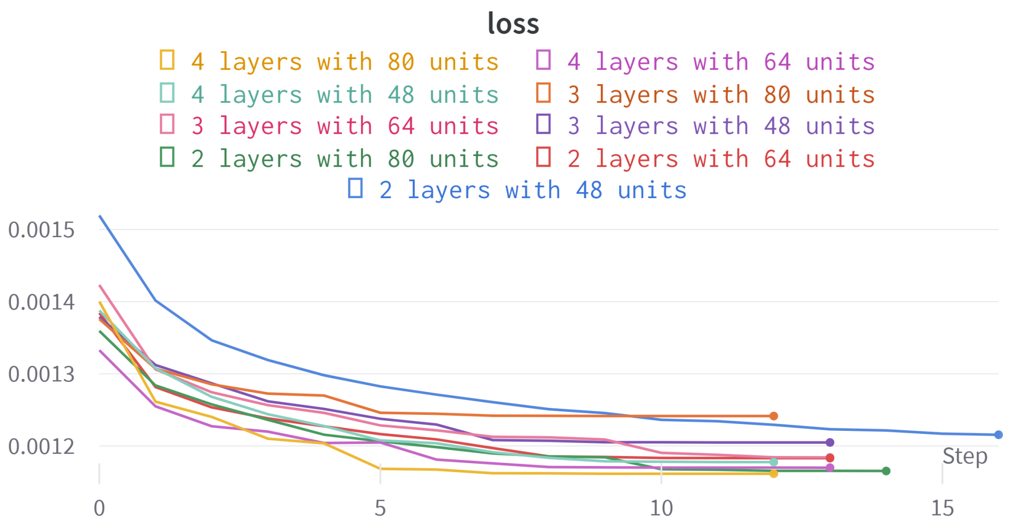
5.9 Neural Network

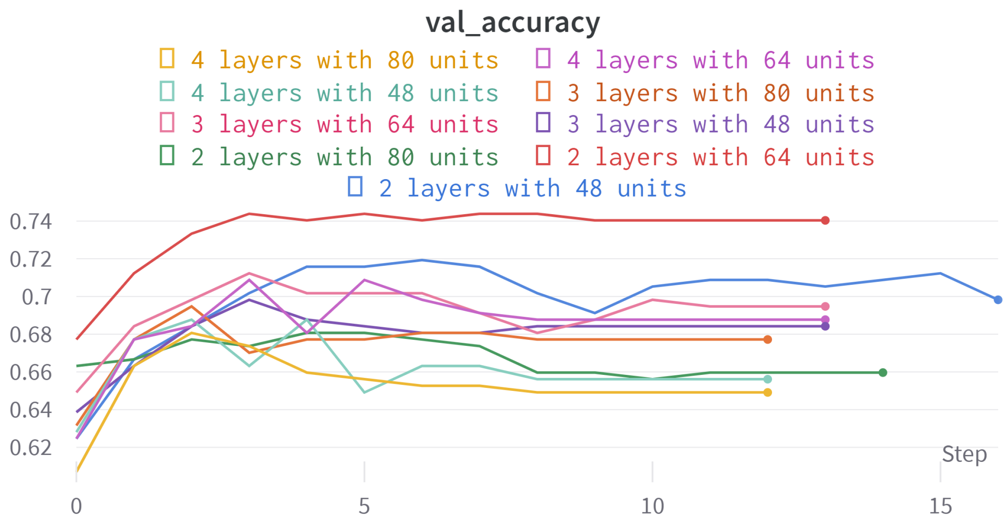
5.9.1 selection of layers and units

Intuitively, we think that the dataset is not that large and we use pca to decrease the feature, therefore, we try layers from 2-4 and units 48, 64 and 80 with active function "tanh" to find better combinations between layers and units. By comparing the training loss, training accuracy, validation loss, validation accuracy:


val_loss
4 layers with 80 units
4 layers with 64 units
4 layers with 48 units
3 layers with 80 units
3 layers with 64 units
3 layers with 48 units
2 layers with 80 units
2 layers with 64 units
2 layers with 48 units
0.68
0.66
0.64
0.62
0.6
0.58
0.56
Step
0
5
10
15


accuracy
4 layers with 80 units
4 layers with 64 units
4 layers with 48 units
3 layers with 80 units
3 layers with 64 units
3 layers with 48 units
2 layers with 80 units
2 layers with 64 units
2 layers with 48 units
0.7
0.65
0.6
0.55
Step
0
5
10
15


loss
4 layers with 80 units
4 layers with 64 units
4 layers with 48 units
3 layers with 80 units
3 layers with 64 units
3 layers with 48 units
2 layers with 80 units
2 layers with 64 units
2 layers with 48 units
0.0015
0.0014
0.0013
0.0012
Step
0
5
10
15



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model = Sequential()
N_HIDDEN = 64
# Add an input shape! (features,)
model.add(Dense(N_HIDDEN, input_shape=(X.shape[1],),
activation='tanh'))
model.add(Dense(N_HIDDEN, activation='tanh'))
model.add(Dense(N_HIDDEN, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

2 layers with 64 units validation loss decreases fastest and has the best performance in validation accuracy. In the rest two parts, it also outperforms. Followed with it is 3 layers with 64 units and 4 layers with 80 units so we use this three to do further adjusting.

5.9.2 Selection of optimizer

Next, we need to choose the best optimizer in finding the best parameters. We find some comments in the web to first learn the difference between optimizers:

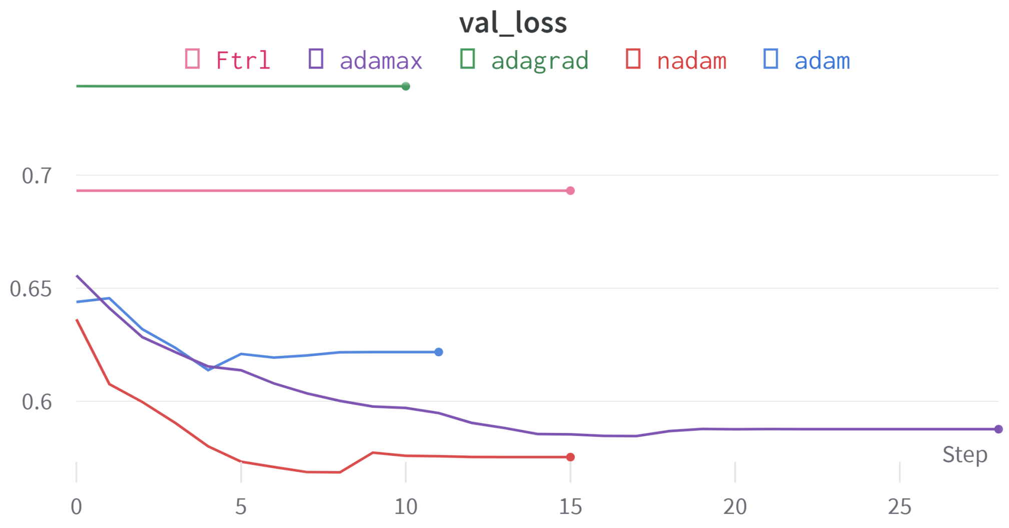
AdaGrad penalizes the learning rate too harshly for parameters which are frequently updated and gives more learning rate to sparse parameters, parameters that are not updated as frequently. In several problems often the most critical information is present in the data that is not as frequent but sparse. So if the problem you are working on deals with sparse data such as tf-idf, etc. Adagrad can be useful.

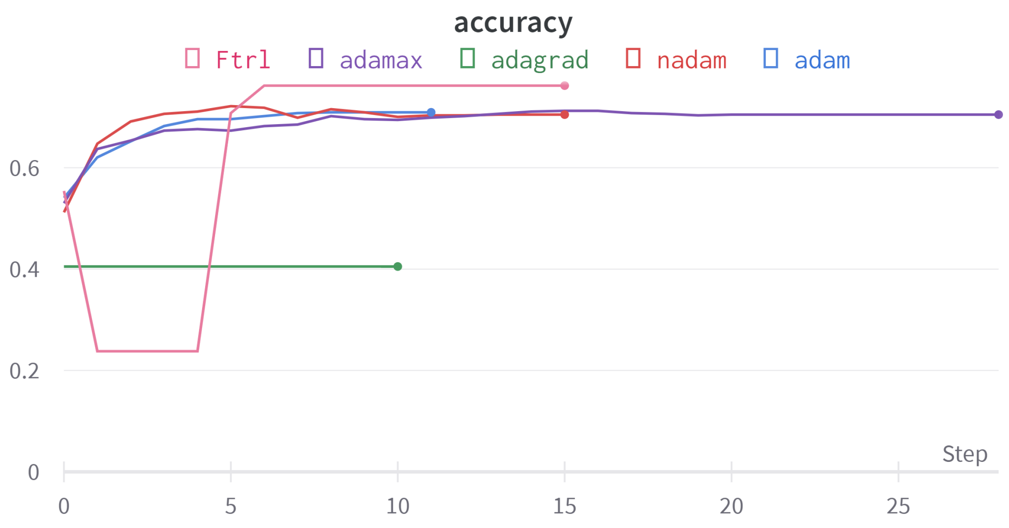
AdaDelta, RMSProp almost works on similar lines with the only difference in Adadelta you don't require an initial learning rate constant to start with.

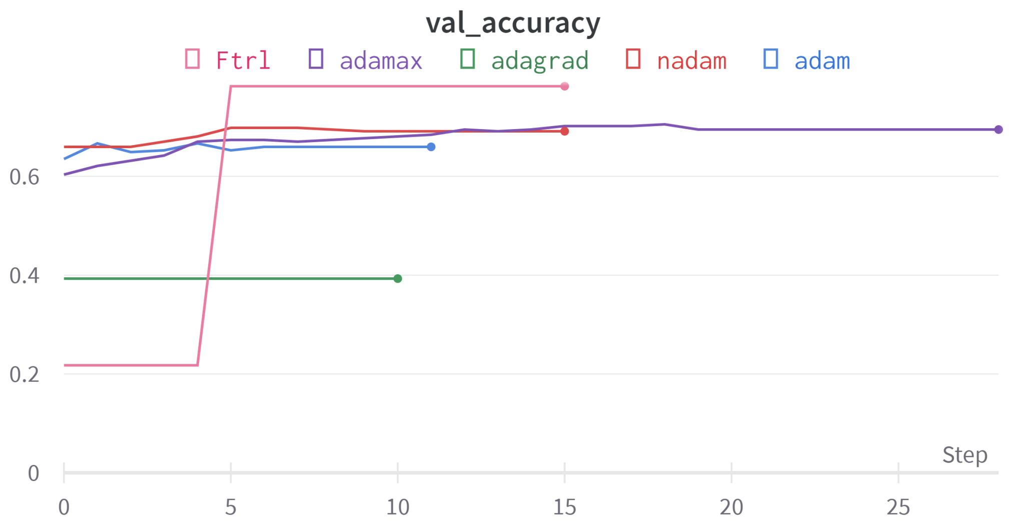
Adam combines the good properties of Adadelta and RMSprop and hence tend to do better for most of the problems.

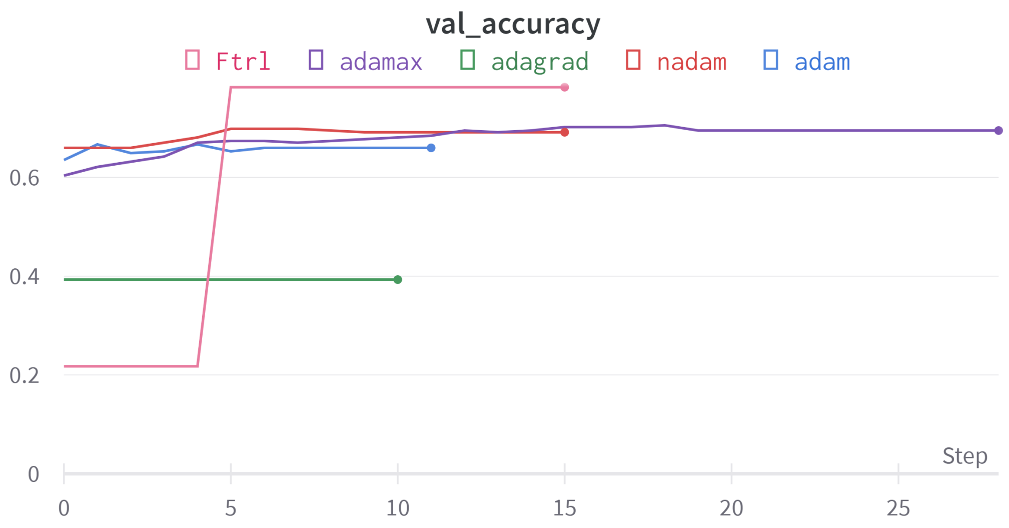
Stochastic gradient descent is very basic and is seldom used now. One problem is with the global learning rate associated with the same. Hence it doesn't work well when the parameters are in different scales since a low learning rate will make the learning slow while a large learning rate might lead to oscillations. Also Stochastic gradient descent generally has a hard time escaping the saddle points. Adagrad, Adadelta, RMSprop, and ADAM generally handle saddle points better. SGD with momentum renders some speed to the optimization and also helps escape local minima better.

Moreover, kerns provides Nadam, adamax and Ftrl with build in functions, we also try them on based on 2 layers and 64 units and get the following results:




accuracy
Ftrl
adamax
adagrad
nadam
adam
0.6
0.4
0.2
0
Step
0
5
10
15
20
25


val_accuracy
Ftrl
adamax
adagrad
nadam
adam
0.6
0.4
0.2
0
Step
0
5
10
15
20
25


val_accuracy
Ftrl
adamax
adagrad
nadam
adam
0.6
0.4
0.2
0
Step
0
5
10
15
20
25

```
# compile the model
# opt = tf.keras.optimizers.Adam(learning_rate=0.001)
# opt = tf.keras.optimizers.Nadam(learning_rate=0.001)
# opt = tf.keras.optimizers.Adagrad(learning_rate=0.001)
# opt = tf.keras.optimizers.Adamax(learning_rate=0.001)
# opt = tf.keras.optimizers.Ftrl(learning_rate=0.001)
model.compile(optimizer = opt,
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

So we choose to use Adamax and Nadam.

5.9.3 Select the callbacks, validation split rate and activation function

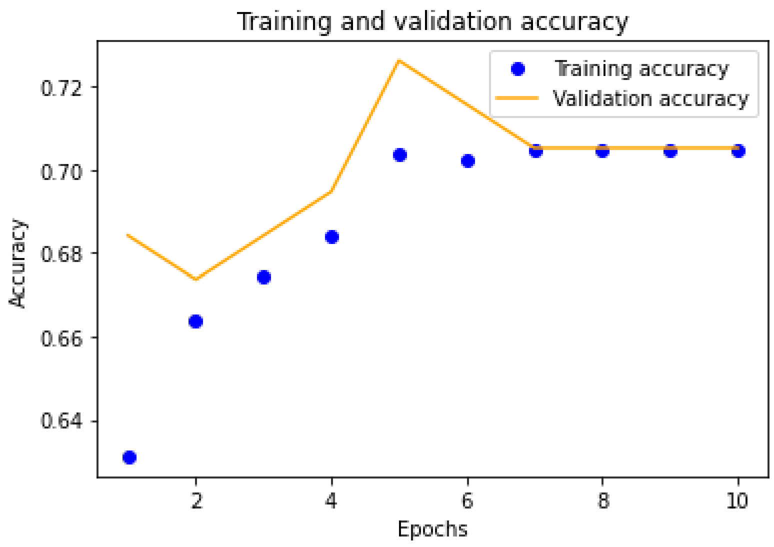
As for the callbacks, we choose to use (1) EarlyStopping and (2) ReduceLROnPlateau since this two give the strong power towards the fitting:

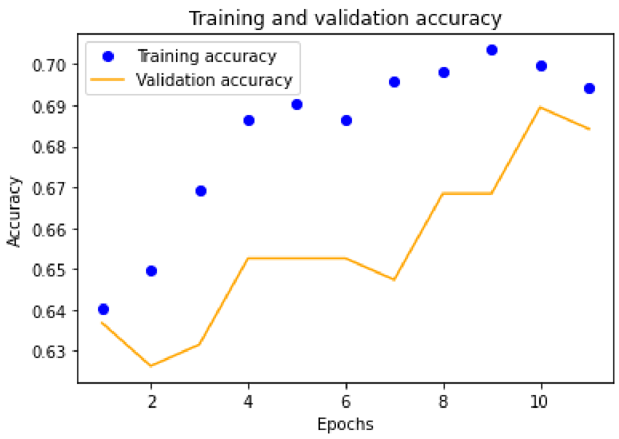
(1) EarlyStopping: Stop training when a monitored metric has stopped improving. In our model, it can reduce the training time largely without worse the prediction results.

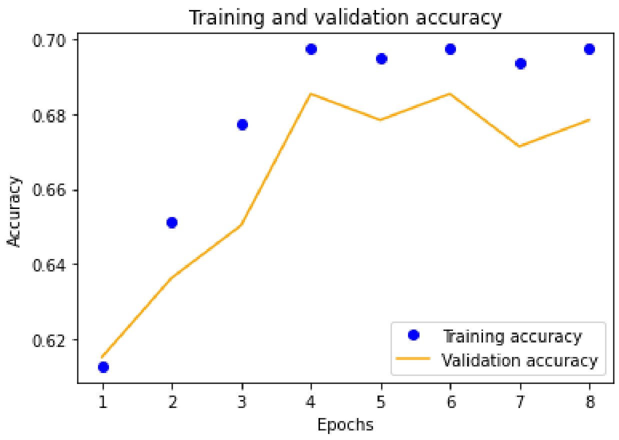
(2) ReduceLROnPlateau: Reduce learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

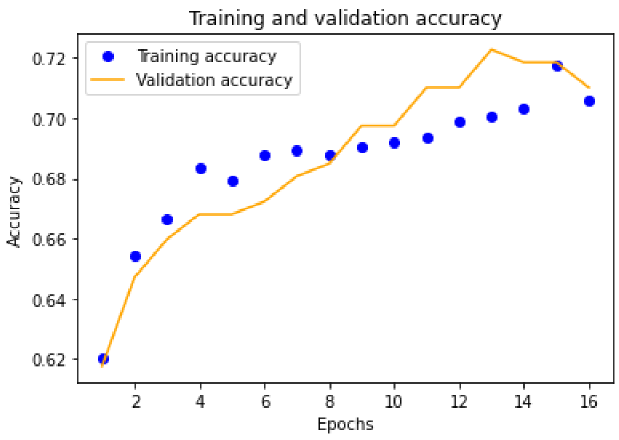
```
es_val_loss = EarlyStopping(monitor='val_loss',patience=10)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
                              patience=2, verbose=1, mode='auto',
                              min_delta=0.0001, cooldown=0, min_lr=1e-
8)
```

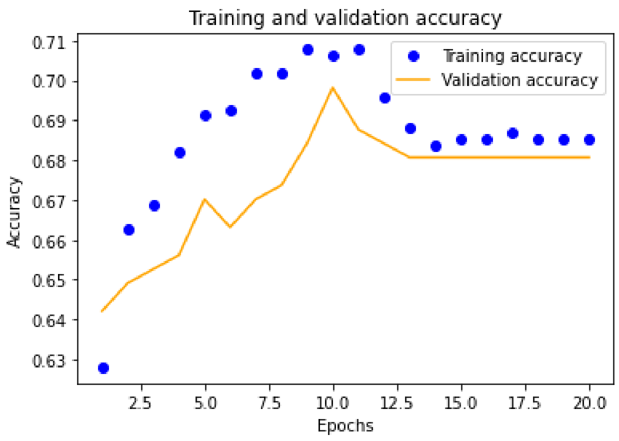
As for the validation split rate, in experience, people always set it in a range of (0.1, 0.3). Let's see the training and validation accuracy on [0.1, 0.15, 0.2, 0.25, 0.3]:


Training and validation accuracy
Training accuracy
Validation accuracy
Accuracy
Epochs
0.72
0.70
0.68
0.66
0.64
2
4
6
8
10


Training and validation accuracy
Training accuracy
Validation accuracy
Accuracy
Epochs
0.70
0.69
0.68
0.67
0.66
0.65
0.64
0.63
2
4
6
8
10

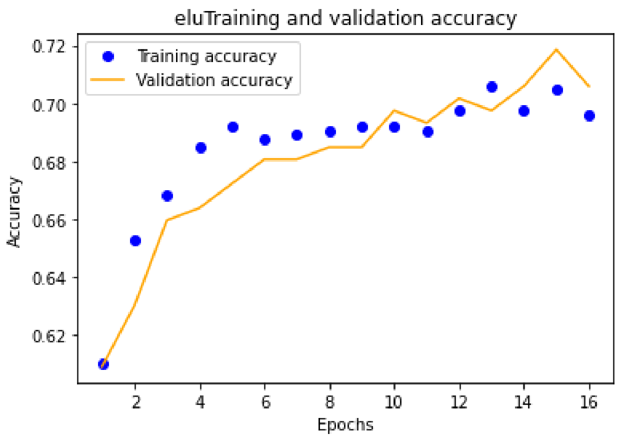

Training and validation accuracy
Training accuracy
Validation accuracy
Accuracy
Epochs
0.70
0.68
0.66
0.64
0.62
1
2
3
4
5
6
7
8

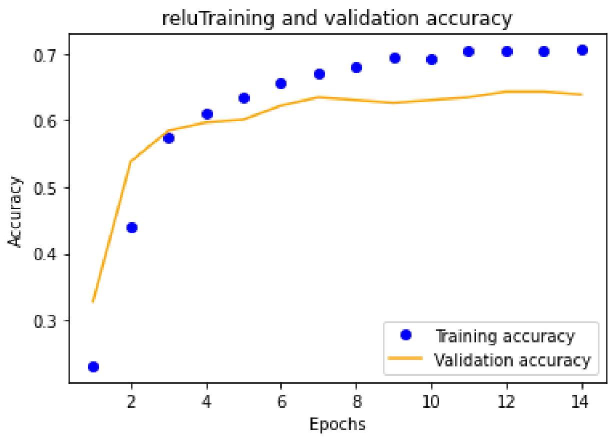


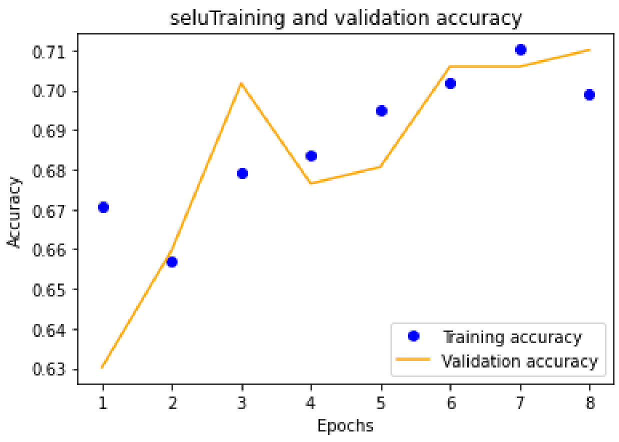


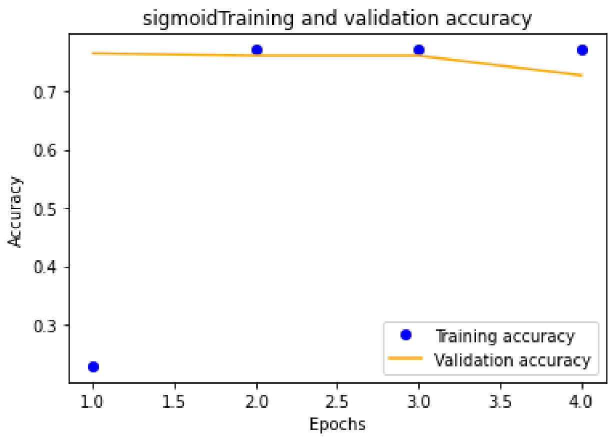
All of them have a good performance in the training set, you can pick whatever you want from set (0.1, 0.3). We use 0.3 here.

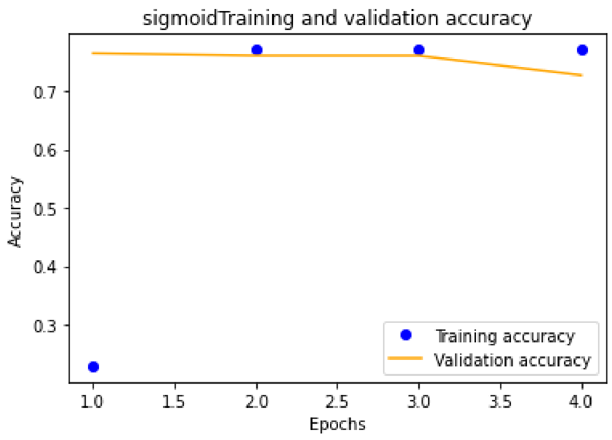
As for the activation function, we try `elu, relu, selu, sigmoid, softsign, tanh`. We set training and validation accuracy as the criteria to choose an appropriate activation function. Here are the results:

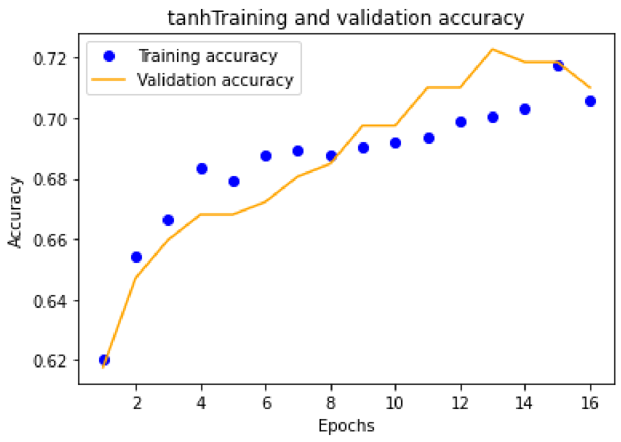



reluTraining and validation accuracy
Accuracy
Epochs
Training accuracy
Validation accuracy
0.3
0.4
0.5
0.6
0.7
2
4
6
8
10
12
14


seluTraining and validation accuracy
Accuracy
Epochs
Training accuracy
Validation accuracy
0.63
0.64
0.65
0.66
0.67
0.68
0.69
0.70
0.71
1
2
3
4
5
6
7
8


sigmoidTraining and validation accuracy
Accuracy
Epochs
Training accuracy
Validation accuracy
0.3
0.4
0.5
0.6
0.7
1.0
1.5
2.0
2.5
3.0
3.5
4.0





According to the graphs, we can see `elu, tanh` perform the best. Therefore, we choose `tanh` function as our activation function.

5.9.4 Select the final parameters

After adding the weight of each class, we finally can train this nn model:

```
counts = np.bincount(y)
weight_for_0 = 1.0 / counts[0]
weight_for_1 = 1.0 / counts[1]
class_weight = {0: weight_for_0, 1: weight_for_1}
history = model.fit(X,
                    y,
                    callbacks=[WandbCallback(), reduce_lr,
es_val_acc],
                    epochs=100, # maximum of epoches.
                    batch_size=128,
                    validation_split=0.3,
                    shuffle=True,
                    class_weight = class_weight,
                    verbose=1)
```

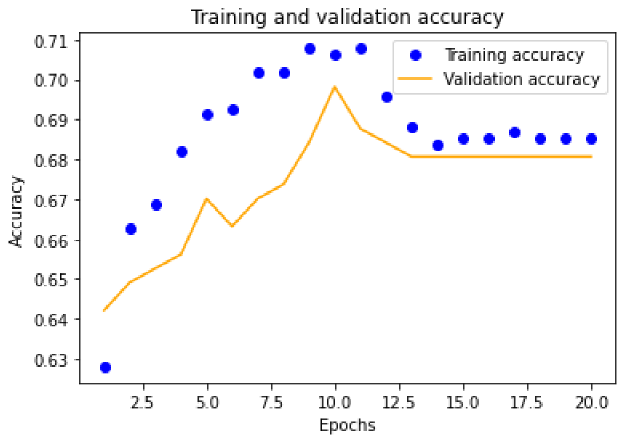
Layers	Units	Optimizer	AUC	F1-score	Precision	Recall	Accuracy
2	64	Nadam	0.7737	0.76	0.79	0.73	0.70
				0.58	0.54	0.63	
3	64	Nadam	0.7898	0.81	0.80	0.82	0.74
				0.61	0.59	0.62	
2	64	Adamax	0.7627	0.79	0.82	0.76	0.73
				0.62	0.59	0.67	
3	64	Adamax	0.7937	0.81	0.81	0.81	0.75
				0.63	0.63	0.63	
4	80	Nadam	0.7649	0.81	0.80	0.83	0.75
				0.61	0.64	0.59	
4	80	Adamax	0.7975	0.82	0.84	0.80	0.77
				0.67	0.64	0.71	

Finally, we select 4 layers, 80 units, `tanh` activation function, validation rate = 0.3 and Adamax optimizer.

5.9.5 Add regularization (set seed 80)

Initial results :

AUC	F1-score	Precision	Recall	Accuracy
0.7982	0.83	0.83	0.83	0.77
	0.67	0.67	0.67	



5.9.6 L1 and L2 regularization

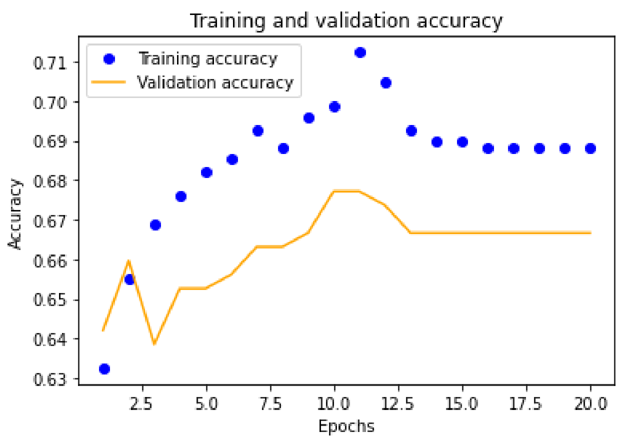
The layers expose 3 keyword arguments:

- `kernel_regularizer`: Regularizer to apply a penalty on the layer's kernel
- `bias_regularizer`: Regularizer to apply a penalty on the layer's bias
- `activity_regularizer`: Regularizer to apply a penalty on the layer's output

We firstly add regularization in the first layer:

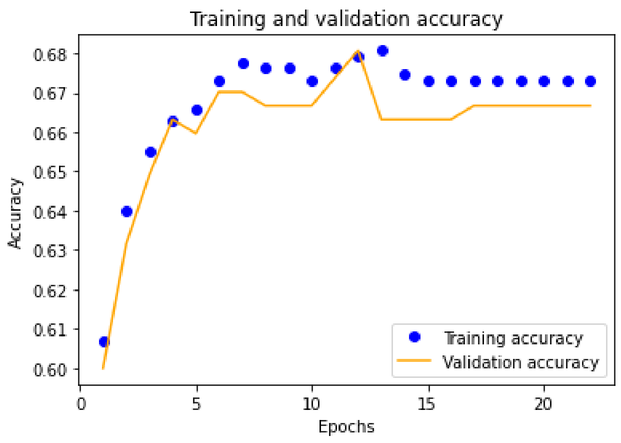
```
model.add(Dense(N_HIDDEN, input_shape=(X.shape[1],),
activation='tanh',
                kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-
4),
                bias_regularizer=regularizers.l2(1e-4),
                activity_regularizer=regularizers.l2(1e-5)))
```

AUC	F1-score	Precision	Recall	Accuracy
0.7913	0.82	0.81	0.82	0.75
	0.63	0.64	0.63	



Then we add regularization in all layers:

AUC	F1-score	Precision	Recall	Accuracy
0.7616	0.77	0.79	0.75	0.70
	0.58	0.55	0.61	



After adding the regularization in all layers, it seems that the training accuracy is closer to the validation accuracy, however, the model fitting becomes worse. Maybe the reason is that the use of early stopping leads to extreme regularization. Then we try to delete the early stopping term to see the result.

AUC	F1-score	Precision	Recall	Accuracy
0.7620	0.75	0.79	0.72	0.69
	0.58	0.53	0.63	

The results are still bad, which means that probability datasets is so small that it is not suitable by L1 and L2 regularizations. And moreover, we have used the PCA to do the feature selections and in the above training and validation accuracy graph, we also see that there is no overfit problem, therefore, L1 and L2 may not work. Let's try dropout method to do further findings.

5.9.7 Dropout

We try to dropout 25% units in each layers to see the results:

AUC	F1-score	Precision	Recall	Accuracy
0.7990	0.83	0.84	0.81	0.77
	0.68	0.65	0.71	

It seems that there is little improvement in the results. Let's try 20% and 30% dropout rate:

AUC	F1-score	Precision	Recall	Accuracy
0.8	0.83	0.80	0.82	0.77
	0.64	0.69	0.66	

AUC	F1-score	Precision	Recall	Accuracy
0.7998	0.82	0.83	0.81	0.77
	0.65	0.64	0.67	

The results show that whether we use dropout method, the accuracy will not increase with a decrease on F1-score. Combined with L1 and L2 regularization, the model may no need further regularizations. And the predicting results in the training set says:

AUC	F1-score	Precision	Recall	Accuracy
0.7713	0.78	0.88	0.70	0.70
	0.52	0.41	0.69	

6 Conclusion

6.1 Comparison of Models

1. First, for linear models: last square, LDA, logistic regression, softmax, the data is not well linearly separable the data has a small overlap between the two categories, which is difficult to distinguish by linearity and simple nonlinearity (regularization). The results of logistic regression and softmax regression are theoretically the same, but the learning rate, number of iterations, and termination conditions are different, resulting in slightly different results. For LDA, the model performs best for the data set with normal distribution, and after PCA, the data basically conforms to the normal distribution, so it stands out among the linear models.
2. Secondly, for the least square & gaussian kernel, the linear model is converted to nonlinear by the transformation of the kernel function, so the f1-score has a significant improvement, but for the AUC, we can see from the ROC picture that it has a high AUC for some thresholds, but there are also thresholds that make it perform badly. In addition, since the mean and variance are randomly generated, it also leads to the instability of the model.
3. Then, for the KNN model, although it is also nonlinear, KNN has a good performance with a large number of samples and a small number of features (the number of samples is much larger than the number of features). For the dataset of this task, the sample size is relatively small and the number of features is too large, so the KNN model does not perform well.
4. For the nn model, it can accurately grasp the nonlinear classification boundary, so the result is significantly better than the linear classifier. However, due to the limited information provided by the data, the real nonlinear boundary is not fully explored, so the effect cannot reach the peak.
5. For the tree model, the amount of data for this task is limited, and it is a problem of exploring the nonlinear boundary, the tree model is a better choice. But for single decision tree, there still appears the problem of too large variance, so the random forest model combines the strength of single tree and

makes up for the disadvantage of too large variance, so the random forest model performs the best among the nine models.

6.2 Pros and Cons of each Model

1. KNN

   Pros: can fit nonlinear classification boundary data

   Cons: not very good performance for data with few dimensions

2. Least squares classification.

   Pros: fast, high interpretability

   Cons: It can only be a linear classifier and is easily affected by outliers

3. Least squares classification with Gaussian basis functions.

   Pros: compared to Least squares classification can be generated to do non-linear classification boundary

   Cons: There is a certain randomness in generating basis functions

4. LDA

   Pros: if the data is Gaussian distributed, the classification effect is higher than other linear classifiers

   Cons: can only do linear classifier

5. Logistic regression with/without regularization (L2) \ Softmax regression with/without regularization (L2)

   Pros: if the data is not Gaussian distributed, the classification effect is higher than other linear classifiers; and regularization can be added

   Cons: can only do linear classifier

6. Decision tree

   Pros: can generate nonlinear classification boundaries and is not affected by data dimensionality

   Cons: the results have large variance

7. Random forest

   Pros: can generate non-linear classification boundaries, and is not affected by data dimensionality, with small variance

   Cons: long training time

8. Neural networks

   Pros: can fit all kinds of classification boundaries

   Cons: requires a lot of information from the training set

7 Appendix

7.1 References:

[1] Source Data: https://archive.ics.uci.edu/ml/datasets/Polish+companies+bankruptcy+data

[2] Prove and application of SoftMax method: https://zhuanlan.zhihu.com/p/98061179

7.2 Code of Models

7.2.1 Code for KNN

```
import pandas as pd
import numpy as np
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score, precision_score,
recall_score, f1_score, roc_curve
import matplotlib.pyplot as plt
import time

class knn:
    def __init__(self,k):
        assert 1<=k
        self.k = k
        self.X = None
        self.y = None

    def fit(self,train_X,train_y):
        self.X = train_X
        self.y = train_y

    def predict(self,test_X):
        n = test_X.shape[0]
        result = np.ones(n)
        for i in range(n):
            new_X = test_X.iloc[i,:]
            nn = (np.argsort(((self.X - new_X)**2).sum(1)))[:self.k]
            result[i] = self.y.iloc[nn,].mean()
        return result


#%% Read the data
# train = pd.read_csv("..\\bankruptcy data\\adj_imp_train.csv")
# test = pd.read_csv("..\\bankruptcy data\\adj_imp_test.csv")
# train = pd.read_csv("..\\bankruptcy
data\\training_knn.csv",index_col=0)
# test = pd.read_csv("..\\bankruptcy
data\\testing_knn.csv",index_col=0)
# train = pd.read_csv("..\\bankruptcy data\\smote_train.csv")
# test = pd.read_csv("..\\bankruptcy data\\adj_imp_test.csv")

train = pd.read_csv("..\\bankruptcy data\\pca_train.csv")
test = pd.read_csv("..\\bankruptcy data\\pca_test.csv")

train_X = train.iloc[:,:-1]
train_y = train.iloc[:,-1]
test_X = test.iloc[:,:-1]
test_y = test.iloc[:,-1]

#%% Change the hyperparameter

auc_list=[]
precision_list=[]
recall_list=[]
f1score_list=[]
for i in range(1,30):
    model=knn(k=i)
    model.fit(train_X,train_y)
    score=model.predict(test_X)
    predict = score >= 0.33
    auc=roc_auc_score(test_y, score)
    precision = precision_score(test_y, predict)
    recall = recall_score(test_y,predict)
    f1score = f1_score(test_y,predict)
    auc_list.append(auc)
    precision_list.append(precision)
    recall_list.append(recall)
    f1score_list.append(f1score)


plt.plot(range(1,30),auc_list,label="auc")
plt.plot(range(1,30),precision_list,label="precision")
plt.plot(range(1,30),recall_list,label="recall")
plt.plot(range(1,30),f1score_list,label="f1 score")
plt.legend()
plt.xlabel('Numbers of Neighbors')
plt.ylabel('indicator')

#%% Model report test set
tic = time.time()
model = knn(k=16)
# model = knn(k=12)
model.fit(train_X,train_y)

score = model.predict(test_X)
toc = time.time()
auc = roc_auc_score(test_y, score)



fpr, tpr, _ = roc_curve(test_y, score)

plt.figure(figsize=(12, 6))
plt.subplot(121)
plt.plot(fpr, tpr, color='darkorange',
         label='ROC curve (area = %0.2f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC for testing set')
plt.legend(loc="lower right")


print("auc:", auc)

print(classification_report(test_y, predict>0.3))

#%% Model report train set
model = knn(k=16)
# model = knn(k=12)
model.fit(train_X,train_y)

score = model.predict(train_X)

auc = roc_auc_score(train_y, score)

fpr, tpr, _ = roc_curve(train_y, score)

plt.subplot(122)
plt.plot(fpr, tpr, color='darkorange',
         label='ROC curve (area = %0.2f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC for training set')
plt.legend(loc="lower right")
plt.show()

print("auc:", auc)
print(classification_report(test_y, predict>0.33))
```

7.2.2 Code for LS

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import
classification_report,confusion_matrix,roc_auc_score, roc_curve

class LeastSquare():
    def __init__(self, train_address = '../data/pca_train.csv',
test_address = '../data/pca_test.csv'):
        self.train_address = train_address
        self.test_address = test_address
        self.dataLoad()

    def dataLoad(self):
        self.origin_train = pd.read_csv(self.train_address)
        self.origin_test = pd.read_csv(self.test_address)

    def classOneHot(self, label_arr):
        """
        Transform label array into matrix. The size is n*k, k is the
number of classes
        This is one-hot transformation
        """
        n_class = np.unique(label_arr).size
        y_matrix = np.zeros((label_arr.shape[0],n_class))
        y_matrix[label_arr==0,0] = 1
        y_matrix[label_arr==1,1] = 1
        return y_matrix.T

    def leastSquare(self, lam = 0):
        training_set = self.origin_train
        testing_set = self.origin_test
        y_vec = training_set['class']
        x_matrix = training_set.drop(['class'], axis = 1)
        # x_matrix = x_matrix.apply(lambda t:((t - np.mean(t)) /
np.std(t)))
        x_matrix['one'] = 1
        y_vec = self.classOneHot(y_vec)
        x_matrix = np.array(x_matrix)
        x_matrix = x_matrix.T
        part1 = np.dot(y_vec, x_matrix.T)
        part2 = np.linalg.inv(np.dot(x_matrix, x_matrix.T)+
                                lam *
np.diag(np.ones(len(training_set.T))))
        self.beta_hat = np.dot(part1, part2)
        test_data = testing_set.drop(['class'], axis = 1)
        # test_data = test_data.apply(lambda t:((t - np.mean(t)) /
np.std(t)))

        test_data['one'] = 1
        self.y_pred = np.dot(self.beta_hat, test_data.T)

    def testResult(self):
        test = self.origin_test
        prediction = self.y_pred
        prediction_df= pd.DataFrame(prediction.T)
        prediction_df["predict"] = 0
        prediction_df.loc[prediction_df[1] > prediction_df[0],
"predict"] = 1
        target_names = ['class 1', 'class 2']
        self.result_report = classification_report(test['class'],
prediction_df['predict'], target_names=target_names)
        self.confu_matrix = confusion_matrix(test['class'],
prediction_df['predict'])
        test_auc = roc_auc_score(test['class'], prediction_df[1])
        print(confusion_matrix(test['class'],
prediction_df['predict']))
        print(classification_report(test['class'],
prediction_df['predict']))
```

```
        print(roc_auc_score(test['class'], prediction_df[1]))

        score = prediction_df[1]
        test_y = self.origin_test['class']
        auc = roc_auc_score(test_y, score)
        fpr, tpr, _ = roc_curve(test_y, score)
        plt.figure(figsize=(12, 6))
        plt.subplot(121)
        plt.plot(fpr, tpr, color='darkorange',
                label='ROC curve (area = %0.2f)' % test_auc)
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for testing set')
        plt.legend(loc="lower right")

if __name__=="__main__":
    p = LeastSquare()
    p.leastSquare(lam=0)
    p.testResult()
```

7.2.3 Code for LS Gaussian

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score, precision_score,
recall_score, f1_score, roc_curve
from sklearn.metrics import roc_auc_score
class LeastGaussian():
    def __init__(self):
        self.recordDict = {"class1 F1": [], "class2 F1": [],
"Accuracy": [], "AUC": []}
        self.readData()
        self.dataSqrt(1/4)
        self.outsideAdj()
        self.kClusteringMean(20)
        self.normalRandomMean(80, 119020084)
        self.probaLst = []

    def readData(self):
        train_org = pd.read_csv("../data/imp_train.csv")
        self.train_x = train_org.drop(['class'], axis = 1)
        self.origin_y = train_org["class"]
        self.train_y = self.classOneHot(train_org["class"])
        test_org = pd.read_csv("../data/imp_test.csv")
        self.test_x = test_org.drop(['class'], axis = 1)
        self.test_y = test_org["class"]

    def classOneHot(self, label_arr):
        """
        Transform label array into matrix. The size is n*k, k is the
number of classes
        This is one-hot transformation
        """
        n_class = np.unique(label_arr).size
        y_matrix = np.zeros((label_arr.shape[0],n_class))
        y_matrix[label_arr==0,0] = 1
        y_matrix[label_arr==1,1] = 1
        return y_matrix.T

    def dataSqrt(self, power):
        """
        First transform all the data into positive
        Then take square root
        Lastly, transform the negative data into negative.
        """
        train_x_abs = np.power(np.abs(self.train_x), power)
        train_x_abs[self.train_x<0] = -train_x_abs[self.train_x<0]
        self.train_x = train_x_abs
        test_x_abs = np.power(np.abs(self.test_x), power)
        test_x_abs[self.test_x<0] = -test_x_abs[self.test_x<0]
        self.test_x = test_x_abs


    def outsideAdj(self):
        mu = self.train_x.mean()
        sigma = np.sqrt(self.train_x.var())
        upper_bound = mu + 1 * sigma
        lower_bound = mu - 1 * sigma
        for attr in lower_bound.index:
            self.train_x[attr][self.train_x[attr] > upper_bound[attr]]
= upper_bound[attr]
            self.test_x[attr][self.test_x[attr] > upper_bound[attr]] =
upper_bound[attr]
            self.train_x[attr][self.train_x[attr] < lower_bound[attr]]
= lower_bound[attr]
            self.test_x[attr][self.test_x[attr] < lower_bound[attr]] =
lower_bound[attr]
        self.train_x = (self.train_x - self.train_x.min()) /
(self.train_x.max() - self.train_x.min())*10
        self.test_x = (self.test_x - self.test_x.min()) /
(self.test_x.max() - self.test_x.min())*10

    def kClusteringMean(self, n):
        """
        Use k-clustering method to find 20 mean values
        """
        train_adj = self.train_x.copy()
        train_adj["cluster"] = KMeans(n_clusters=n, max_iter=10000,
random_state=119020).fit_predict(self.train_x)
        train_adj_mean = train_adj.groupby("cluster").mean()
        self.k_cluster_mean = train_adj_mean

    def normalRandomMean(self, n, seed):
        mean_var_lst = list(zip(self.train_x.mean(),
np.sqrt(self.train_x.var())))
        zeros = np.zeros([n, len(mean_var_lst)])
        for attr in range(len(mean_var_lst)):
            np.random.seed(seed)
            zeros[:,attr] = np.random.normal(mean_var_lst[attr][0],
mean_var_lst[attr][1],n)
        zeros = pd.DataFrame(zeros, columns =
self.k_cluster_mean.columns)
        self.mean_generate = pd.concat([pd.DataFrame(zeros),
self.k_cluster_mean])



    def gaussianTransform(self, row):
        gaussian_x_train = np.exp(- (self.train_x -
self.mean_generate.iloc[row,])**2 / 2 * self.train_x.var())
        guassian_x_test = np.exp(- (self.test_x -
self.mean_generate.iloc[row,])**2 / 2 * self.test_x.var())
        gaussian_x_train["constant"] = 1
        guassian_x_test["constant"] = 1
        self.gaussian_x_train = np.array(gaussian_x_train).T
        self.gaussian_x_test = np.array(guassian_x_test).T

    def gaussianTrain(self, row):
        self.gaussianTransform(row)
        T_XT = np.dot(self.train_y, self.gaussian_x_train.T)
        X_XT_inv = np.linalg.pinv(np.dot(self.gaussian_x_train,
self.gaussian_x_train.T))
        self.parameters = np.dot(T_XT, X_XT_inv)
        self.gaussian_predict()

    def gaussian_predict(self, train_test=False):
        df = pd.DataFrame(np.dot(self.parameters,
self.gaussian_x_test)).T
        self.mydf = df
        df["predict"] = 0
        df.loc[df[1]>df[0], "predict"] = 1
        target_names = ['class 1', 'class 2']
        self.auc_score = roc_auc_score(np.array(self.test_y), df[1])
        self.probaLst.append(df[1])
        result = classification_report(np.array(self.test_y),
df['predict'], target_names=target_names)
        self.result = result
        self.recordDict["class1
F1"].append(float(self.result.split("\n")[2].split()[4]))
        self.recordDict["class2
F1"].append(float(self.result.split("\n")[3].split()[4]))

 self.recordDict["Accuracy"].append(float(self.result.split("\n")
[5].split()[1]))
        self.recordDict["AUC"].append(self.auc_score)

    def resultVisualize(self):
        plt.figure(figsize=(12,8))
        plt.style.use("ggplot")
        plt.plot(self.recordDict["class1 F1"], linestyle='--',
color="grey",lw=2)
        plt.plot(self.recordDict["class2 F1"], linestyle='--',
color="darkgrey",lw=2)
        plt.plot(self.recordDict["Accuracy"], linestyle='-',
color="blue", lw=3)
        plt.plot(self.recordDict["AUC"], linestyle='-', color="red",
lw=3)
        plt.axvline(20, ls="--", lw=3)
        plt.legend(self.recordDict.keys(), fontsize=14, loc=1)
        plt.text(2, 0.2, "K-clustering", color = "red", font =
{'family': 'normal', 'weight': 'bold', 'size': 18})
        plt.text(40, 0.2, "Random Generate", color = "red", font =
{'family': 'normal', 'weight': 'bold', 'size': 18})
        plt.xlabel("Times", font = {'family': 'normal', 'weight':
'bold', 'size': 18})
        plt.ylabel("Predict Score", font = {'family': 'normal',
'weight': 'bold', 'size': 18})
```

```
        plt.xticks(font = {'family': 'normal', 'weight': 'normal',
'size': 14})
        plt.yticks(font = {'family': 'normal', 'weight': 'normal',
'size': 14})
        plt.title("LS Classification with Gaussian Basis
Functions\n(Best Accuracy={}, Best AUC=
{})".format(np.array(self.recordDict["Accuracy"]).max(),\
            np.round(np.array(self.recordDict["AUC"]).max(), 2)),\
             font = {'family': 'normal', 'weight': 'bold', 'size':
18})

    def resultReport(self):
        acc_array = np.array(self.recordDict["Accuracy"])
        best_row = np.where(acc_array == acc_array.max())[0][0]
        self.gaussianTrain(best_row)
        print(self.result)
        print("AUC: ", np.round(self.auc_score, 2))
    def draw_roc(self):
        score = self.probaLst[-1]
        score = (score - score.min()) / (score.max() - score.min())
        auc = roc_auc_score(self.test_y, score)
        fpr, tpr, _ = roc_curve(self.test_y, score)
        plt.figure(figsize=(12, 6))
        plt.subplot(121)
        plt.plot(fpr, tpr, color='darkorange',
                label='ROC curve (area = %0.2f)' % auc)
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for testing set')
        plt.legend(loc="lower right")

        self.gaussian_x_test = self.gaussian_x_train
        self.test_y = self.origin_y
        self.gaussian_predict()
        score = self.probaLst[-1]
        score = (score - score.min()) / (score.max() - score.min())
        auc = roc_auc_score(self.origin_y, score)
        fpr, tpr, _ = roc_curve(self.origin_y, score)
        plt.subplot(122)
        plt.plot(fpr, tpr, color='darkorange',
                label='ROC curve (area = %0.2f)' % auc)
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for training set')
        plt.legend(loc="lower right")
        plt.show()

if __name__ == "__main__":
    p = LeastGaussian()
    for i in range(100):
        p.gaussianTrain(i)
    p.resultVisualize()
    print("The prediction result of LS classification with gaussian
basis is:")
    p.resultReport()
    p.draw_roc()
```

7.2.4 Code for LDA

```
import pandas as pd
import numpy as np
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import time

class LDA_2class:
    def __init__(self):
        self.mean = None
        self.std = None
        self.w = None
        self.larger = None

    def fit(self,train_X,train_y):
        train_X0 = train_X[train_y==0]
        train_X1 = train_X[train_y==1]
        n1 = train_X0.shape[0]
        n2 = train_X1.shape[0]
        m1 = train_X0.mean()
        m2 = train_X1.mean()
        S_B = np.outer(m2-m1,m2-m1)
        # S_W = np.zeros(S_B.shape)
        # for i in range(train_X0.shape[0]):
        #     S_W = np.add(np.outer(train_X0.iloc[i,:]-
m1,train_X0.iloc[i,:]-m1),S_W)
        # for i in range(train_X1.shape[0]):
        #     S_W = np.add(np.outer(train_X1.iloc[i,:]-
m2,train_X1.iloc[i,:]-m2),S_W)
        S_W = ((train_X0-m1).T).dot(train_X0-m1) + ((train_X1-
m2).T).dot(train_X1-m2)
        lamb, W = np.linalg.eigh(np.linalg.pinv(S_W).dot(S_B))
        eiglist = [(lamb[i], W[:, i]) for i in range(len(lamb))]
        eiglist = sorted(eiglist, key = lambda x : x[0], reverse =
True)
        self.w = eiglist[0][1]
        self.w0 = (np.dot(self.w, m1) + np.dot(self.w, m2))/2
        self.larger = np.dot(self.w, m2) >= self.w0

    def score(self,test_X):
        if self.larger:
            return test_X.dot(self.w)
        else:
            return 1-test_X.dot(self.w)

    def predict(self,test_X):
        if self.larger:
            return (test_X.dot(self.w)) >= self.w0
        else:
            return (test_X.dot(self.w)) < self.w0

#%% Read the data
# train = pd.read_csv("..\\bankruptcy data\\adj_imp_train.csv")
# test = pd.read_csv("..\\bankruptcy data\\adj_imp_test.csv")
# train = pd.read_csv("..\\bankruptcy
data\\training_knn.csv",index_col=0)
# test = pd.read_csv("..\\bankruptcy
data\\testing_knn.csv",index_col=0)
# train = pd.read_csv("..\\bankruptcy data\\smote_train.csv")
# test = pd.read_csv("..\\bankruptcy data\\adj_imp_test.csv")

train = pd.read_csv("..\\bankruptcy data\\pca_train.csv")
test = pd.read_csv("..\\bankruptcy data\\pca_test.csv")

train_X = train.iloc[:,:-1]/10
train_y = train.iloc[:,-1]
test_X = test.iloc[:,:-1]/10
test_y = test.iloc[:,-1]

tic = time.time()
model = LDA_2class()
model.fit(train_X,train_y)
toc = time.time()



#%% Model report test set
predict = model.predict(test_X)
score = model.score(test_X)

auc = roc_auc_score(test_y, score)

fpr, tpr, _ = roc_curve(test_y, score)
```

```python
plt.figure(figsize=(12, 6))
plt.subplot(121)
plt.plot(fpr, tpr, color='darkorange',
         label='ROC curve (area = %0.2f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC for testing set')
plt.legend(loc="lower right")

print("auc:", auc)

print(classification_report(test_y, predict))



#%% visualization

plt.hist(score[:100],color="red",label="Class 1")
plt.hist(score[100:],color="blue", label="Class 2")
plt.axvline(1-model.w0,linewidth=5,color="black",linestyle='--')
plt.xlabel('Projected Data')
plt.ylabel('Frequency')
plt.legend()
plt.show()

#%% Model report train set
predict = model.predict(train_X)
score = model.score(train_X)

auc = roc_auc_score(train_y, score)

fpr, tpr, _ = roc_curve(train_y, score)

plt.subplot(122)
plt.plot(fpr, tpr, color='darkorange',
         label='ROC curve (area = %0.2f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC for training set')
plt.legend(loc="lower right")
plt.show()

print("auc:", auc)

print(classification_report(train_y, predict))

#%% visualization

plt.hist(score[:728],color="red",label="Class 1")
plt.hist(score[728:],color="blue",label="Class 2")
plt.axvline(1-model.w0,linewidth=5,color="black",linestyle='--')
plt.xlabel('Numbers of Neighbors')
plt.ylabel('AUC')
plt.legend()
plt.show()
```

7.2.5 Code for Logistic Regression

```
import pandas as pd
import numpy as np
import copy
import matplotlib.pyplot as plt
from sklearn.metrics import
classification_report,confusion_matrix,roc_auc_score, roc_curve



class LogisticRegression():
    def __init__(self, train_address = '../data/pca_train.csv',
test_address = '../data/pca_test.csv'):
        self.lam = None
        self.lr = None
        self.train_address = train_address
        self.test_address = test_address
        self.dataLoad()

    def dataLoad(self):
        self.origin_train = pd.read_csv(self.train_address)
        self.origin_test = pd.read_csv(self.test_address)

    def sigmoid(self, x):
        return 1/(1+np.exp(-x))

    def square_loss(self, y_pred, target):
        return np.mean(pow((y_pred - target),2))

    def classOneHot(self, label_arr):
        """
        Transform label array into matrix. The size is n*k, k is the
number of classes
        This is one-hot transformation
        """
        n_class = np.unique(label_arr).size
        y_matrix = np.zeros((label_arr.shape[0],n_class))
        y_matrix[label_arr==0,0] = 1
        y_matrix[label_arr==1,1] = 1
        return y_matrix.T

    def leastSquare(self, lam = 0, lr = 0.03, iteration = 10000):
        self.lam = lam
        self.lr = lr
        y_tr = self.origin_train['class']
        X_tr = self.origin_train.drop(['class'], axis = 1)
        X_tr = X_tr.apply(lambda t:((t - np.mean(t)) / np.std(t)))
        self.y_te = self.origin_test['class']
        self.X_te = self.origin_test.drop(['class'], axis = 1)
        self.X_te = self.X_te.apply(lambda t:((t - np.mean(t)) /
np.std(t)))
        self.lr = 0.003
        W = np.random.uniform(0,1,len(X_tr.T))
        b = 0.1
        z = np.dot(X_tr, W) + b
        y_pred = self.sigmoid(z)

        self.loss = []
        for i in range(iteration):
            gradient_W = (np.dot((y_pred-y_tr).T, X_tr) + self.lam *
W)/(X_tr.shape[0])
            gradient_b = np.mean(y_pred-y_tr)
            W = W - self.lr * gradient_W
            b = b - self.lr * gradient_b
            z = np.dot(X_tr, W) + b
            y_pred = self.sigmoid(z)
            self.loss.append(self.square_loss(y_pred, y_tr))
        test_z = np.dot(self.X_te, W) + b

        self.prediction = self.sigmoid(test_z)
        self.score = copy.copy(self.prediction)

        self.prediction[np.where(self.prediction<0.5)] = 0
        self.prediction[np.where(self.prediction>=0.5)] = 1

    def testResult(self):
        target_names = ['class 1', 'class 2']
        result_report = classification_report(self.y_te,
self.prediction, target_names=target_names)
        confu_matrix = confusion_matrix(self.y_te, self.prediction)
        auc_score = roc_auc_score(self.y_te, self.score)
        print(self.lam)
        print(confu_matrix)
        print(result_report)
        print(auc_score)
        print(self.loss[-5:])

        score = self.score
        test_y = self.origin_test['class']
        auc = roc_auc_score(test_y, score)
        fpr, tpr, _ = roc_curve(test_y, score)
        plt.figure(figsize=(12, 6))
        plt.subplot(121)
        plt.plot(fpr, tpr, color='darkorange',
                label='ROC curve (area = %0.2f)' % auc_score)
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for testing set')
        plt.legend(loc="lower right")

if __name__=="__main__":
    p = LogisticRegression()
    p.leastSquare(lam=0)
    p.testResult()
```

7.2.6 Code for SoftMax

```
import numpy as np
import pandas as pd
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score, precision_score,
recall_score, f1_score, roc_curve
from sklearn.metrics import roc_auc_score
class Softmax():
    def __init__(self):
        np.random.seed(119020)
        self.dataLoad()
        self.n_class = np.unique(self.train_y).size  # Find how many
classes there are.
        self.n_sample = self.train_x.shape[0]
        self.n_attr = self.train_x.shape[1]
        self.cost_list = []
        self.auc_list = []
        self.train_test = False

    def dataLoad(self):
        origin_train = pd.read_csv("../data/adj_imp_train.csv")
        origin_test = pd.read_csv("../data/adj_imp_test.csv")
        origin_file = pd.concat([origin_train, origin_test], axis=0)
        origin_file["constant"] = 1
        total_x = np.array(origin_file.drop(columns=['class']))
        total_x = preprocessing.normalize(total_x, norm="l2")
        total_y = np.array(origin_file['class'])
        self.train_x = total_x[:origin_train.shape[0]]
        self.test_x = total_x[origin_train.shape[0]:]
        self.train_y = total_y[:origin_train.shape[0]]
        self.test_y = total_y[origin_train.shape[0]:]

    def softMax(self, data, parameter):
        """
        Calculate the classification probability of each class for the
given data
        Return the probability martix, soft_max matrix
        Row sum is 1
        """
        product_value = np.exp(np.dot(data, parameter.T))  # calculate
the sum(w_i*x_i) for each data point and each model
        line_sum =
product_value.sum(axis=1).reshape((product_value.shape[0], -1))  # Sum
each row
        soft_max = product_value / line_sum  # Calculate probability
        return soft_max
    def classOneHot(self, label_arr):
        """
        Transform label array into matrix. The size is n*k, k is the
number of classes
        This is one-hot transformation
        """
        n_class = np.unique(label_arr).size
        y_matrix = np.zeros((label_arr.shape[0],n_class))
        y_matrix[label_arr==0,0] = 1
        y_matrix[label_arr==1,1] = 1
        return y_matrix

    def softMaxTrain(self, alpha=5, iteration=1000):
        self.parameters = np.random.rand(self.n_class, self.n_attr)  #
shape: (2, 64). Initail all the parameters. Each row represents the
vector of parameters. 2 rows because there are two models for
different type y
        y_one_hot = self.classOneHot(self.train_y)
        w_gradient = 0
        for i in range(iteration):
            self.parameters -= alpha * w_gradient
            self.predict()
            soft_max = self.softMax(self.train_x, self.parameters)
            cost = -1/self.n_sample *(y_one_hot *
np.log(soft_max)).sum()  # Calculate cost function
            w_gradient = -1/self.n_sample * np.dot((y_one_hot -
soft_max).T, self.train_x)  # Calculate the gradient
            if np.abs(w_gradient.min()) < 2*10e-5:
                break
            self.w = w_gradient
            self.cost_list.append(cost)
        # self.parameters = init_parameter

    def lossVisual(self):
        plt.figure(figsize = (8,6))
        plt.plot(np.arange(len(self.cost_list)), self.cost_list)
        plt.title("Development of loss during training", \
            fontdict={"family": "normal", "weight": "bold", "size":
16})
        plt.xlabel("Number of iterations", fontdict={"family":
"normal", "weight": "bold", "size": 14})
        plt.ylabel("Loss", fontdict={"family": "normal", "weight":
"bold", "size": 14})
        plt.show()

    def predict(self, train_test=False):
        if train_test:
            self.test_x = self.train_x
            self.test_y = self.train_y
        calculate_result = pd.DataFrame(self.softMax(self.test_x,
self.parameters))
        calculate_result["predict"] = 0
        calculate_result.loc[calculate_result[1] >
calculate_result[0], "predict"] = 1
        self.predict_result = calculate_result["predict"]
        self.classification_prob = calculate_result[1]
        predict_auc = roc_auc_score(np.array(self.test_y),
calculate_result[1])
        self.cal_result = calculate_result
        self.auc_list.append(predict_auc)
        # print(calculate_result)

    def resultAnalysis(self):
        plt.figure(figsize = (8,6))
        target_names = ['class 1', 'class 2']
        if self.train_test:
            print("*****Performance of model on training data*****")
        print("The result of SoftMax prediction is: ")
        result = classification_report(self.test_y,
self.predict_result, target_names=target_names)
        print(result)
        print("AUC is: ", np.round(self.auc_list[-1], 4))
        if not self.train_test:
            plt.plot(self.auc_list)
            plt.xlabel("Number of iterations", fontdict={"family":
"normal", "weight": "bold", "size": 14})
            plt.ylabel("Predict AUC", fontdict={"family": "normal",
"weight": "bold", "size": 14})
            plt.title("Development of prediction AUC during training",
fontdict={"family": "normal", "weight": "bold", "size": 18})
            plt.show()
        print("*****END*****")

    def draw_roc(self):
        score = self.classification_prob
        auc = roc_auc_score(self.test_y, score)
        fpr, tpr, _ = roc_curve(self.test_y, score)
        plt.figure(figsize=(12, 6))
        plt.subplot(121)
        plt.plot(fpr, tpr, color='darkorange',
                label='ROC curve (area = %0.2f)' % auc)
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for testing set')
        plt.legend(loc="lower right")

        self.predict(True)
        score = self.classification_prob
        print(self.train_y.shape)
        print(score.shape)
        auc = roc_auc_score(self.train_y, score)
        fpr, tpr, _ = roc_curve(self.train_y, score)
        plt.subplot(122)
        plt.plot(fpr, tpr, color='darkorange',
                label='ROC curve (area = %0.2f)' % auc)
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for training set')
        plt.legend(loc="lower right")
        plt.show()

if __name__=="__main__":
    p = Softmax()
    p.softMaxTrain(0.01, 50000)
    p.lossVisual()
    p.predict()
    p.resultAnalysis()
    p.predict()
    p.resultAnalysis()
    p.draw_roc()
```

7.2.7 Code for Decision Tree

```
import pybaobabdt
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score, precision_score,
recall_score, f1_score, roc_curve
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import time

#%% Read the data
train = pd.read_csv("..\\bankruptcy data\\adj_imp_train.csv")
test = pd.read_csv("..\\bankruptcy data\\adj_imp_test.csv")
# train = pd.read_csv("..\\bankruptcy
data\\training_knn.csv",index_col=0)
```

```
# test = pd.read_csv("..\\bankruptcy
data\\testing_knn.csv",index_col=0)
# train = pd.read_csv("..\\bankruptcy data\\smote_train.csv")
# test = pd.read_csv("..\\bankruptcy data\\adj_imp_test.csv")

# train = pd.read_csv("..\\bankruptcy data\\pca_train.csv")
# test = pd.read_csv("..\\bankruptcy data\\pca_test.csv")

train_X = train.iloc[:,:-1]/10
train_y = train.iloc[:,-1]
train_y.apply(str)

test_X = test.iloc[:,:-1]/10
test_y = test.iloc[:,-1]
test_y.apply(str)

#%% Model Buliding
features =list(train.columns)

model =
tree.DecisionTreeClassifier(criterion='entropy',random_state=0,min_sam
ples_leaf=40, max_depth=6)
# model =
tree.DecisionTreeClassifier(criterion='gini',random_state=0,min_sample
s_leaf=37)
tic = time.time()
model.fit(train_X,list(train_y))
toc = time.time()
# fig = plt.figure(dpi=500)
# ax1 = fig.add_subplot(111)
# pybaobabdt.drawTree(model, size=10, dpi=500, features=features[:-1],
colormap='Set1',ax=ax1)
# fig.savefig('C:\\Users\\L\Desktop\\sta\\FTE4560\\project-
1\\report\\Decision Tree.png', format='png', dpi=300,
transparent=True)
#%% Model report test set
# predict = model.predict(test_X)
```

```
tic = time.time()
score = model.predict_proba(test_X)[:,1]
toc = time.time()
predict = score >= 0.3
auc = roc_auc_score(test_y, score)

fpr, tpr, _ = roc_curve(test_y, score)

plt.figure(figsize=(12, 6))
plt.subplot(121)
plt.plot(fpr, tpr, color='darkorange',
         label='ROC curve (area = %0.2f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC for testing set')
plt.legend(loc="lower right")

print("auc:", auc)
print(classification_report(test_y, predict))

#%% Model report train set
# predict = model.predict(train_X)
score = model.predict_proba(train_X)[:,1]
predict = score >= 0.3
auc = roc_auc_score(train_y, score)

fpr, tpr, _ = roc_curve(train_y, score)



plt.subplot(122)
plt.plot(fpr, tpr, color='darkorange',
         label='ROC curve (area = %0.2f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC for training set')
plt.legend(loc="lower right")
plt.show()

print("auc:", auc)
print(classification_report(train_y, predict))



#%% Change the hyperparameter

auc_list=[]
precision_list=[]
recall_list=[]
f1score_list=[]
for i in range(1,20):
    model =
tree.DecisionTreeClassifier(criterion='entropy',random_state=0,max_dep
th=i)
    model.fit(train_X,train_y)
    score = model.predict_proba(test_X)[:,1]
    predict = score >= 0.33
    auc=roc_auc_score(test_y, score)
    precision = precision_score(test_y, predict)
    recall = recall_score(test_y,predict)
    f1score = f1_score(test_y,predict)
    auc_list.append(auc)
    precision_list.append(precision)
    recall_list.append(recall)
    f1score_list.append(f1score)

plt.figure(figsize=(12, 6))
plt.subplot(121)
plt.plot(range(1,20),auc_list,label="auc")
plt.plot(range(1,20),precision_list,label="precision")
plt.plot(range(1,20),recall_list,label="recall")
plt.plot(range(1,20),f1score_list,label="f1 score")
plt.legend()
plt.xlabel('max depth')
plt.title('Entropy')
plt.ylabel('indicator')


auc_list=[]
precision_list=[]
recall_list=[]
f1score_list=[]
for i in range(1,20):
    model =
tree.DecisionTreeClassifier(criterion='gini',random_state=0,max_depth=
i)
    model.fit(train_X,train_y)
    score = model.predict_proba(test_X)[:,1]
    predict = score >= 0.33
    auc=roc_auc_score(test_y, score)
    precision = precision_score(test_y, predict)
    recall = recall_score(test_y,predict)
    f1score = f1_score(test_y,predict)
    auc_list.append(auc)
    precision_list.append(precision)
    recall_list.append(recall)
    f1score_list.append(f1score)

plt.subplot(122)
plt.plot(range(1,20),auc_list,label="auc")
plt.plot(range(1,20),precision_list,label="precision")
plt.plot(range(1,20),recall_list,label="recall")
plt.plot(range(1,20),f1score_list,label="f1 score")
plt.legend()
plt.xlabel('max depth')
plt.title('Gini')
plt.ylabel('indicator')
plt.show()

#%% Change the hyperparameter

auc_list=[]
precision_list=[]
recall_list=[]
f1score_list=[]
for i in range(1,100):
    model =
tree.DecisionTreeClassifier(criterion='entropy',random_state=0,max_dep
th=6,min_samples_leaf=i)
    model.fit(train_X,train_y)
    score = model.predict_proba(test_X)[:,1]
    predict = score >= 0.33
    auc=roc_auc_score(test_y, score)
    precision = precision_score(test_y, predict)
    recall = recall_score(test_y,predict)
    f1score = f1_score(test_y,predict)
    auc_list.append(auc)
    precision_list.append(precision)
    recall_list.append(recall)
    f1score_list.append(f1score)

plt.plot(range(1,100),auc_list,label="auc")
plt.plot(range(1,100),precision_list,label="precision")
plt.plot(range(1,100),recall_list,label="recall")
plt.plot(range(1,100),f1score_list,label="f1 score")
plt.legend()
plt.xlabel('min samples leaf')
plt.ylabel('indicator')
plt.show()
```

7.2.8 Code for Random Forest

```
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
from sklearn.metrics import roc_auc_score
import numpy as np
from sklearn.model_selection import GridSearchCV
import seaborn as sns
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score, precision_score,
recall_score, f1_score, roc_curve
class Randomforest():
    def __init__(self):
        self.dataLoad()
        self.report=False
        self.train_test = False
        self.score_record = {}

    def dataLoad(self):
        train_set = pd.read_csv("../data/imp_train.csv")
        test_set = pd.read_csv("../data/imp_test.csv")
        self.training_data = train_set.drop(columns = ["class",
'Attr14', 'Attr18'])
        self.training_label = train_set["class"]
        self.testing_data = test_set.drop(columns = ["class",
'Attr14', 'Attr18'])
        self.testing_label = test_set["class"]
        with open("../data/features_meaning.txt") as f:
            self.features_meaning = f.readlines()

    def train(self, max_estimators=10):
        param_grid = {'n_estimators': range(1, max_estimators, 1)}
        for feature in ["sqrt", "log2", None]:
            forest_reg = RandomForestClassifier(max_features=feature,
criterion="entropy", random_state=119020)
            grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                                    scoring='roc_auc', n_jobs=3,
return_train_score=True)
            grid_search.fit(self.training_data, self.training_label)
            print(feature, grid_search.best_params_)
            self.score_record[feature] =
pd.DataFrame(grid_search.cv_results_)["mean_test_score"]
            self.test(grid_search.best_params_["n_estimators"],
feature)

    def train_criterion(self, max_estimators=10):
        param_grid = {'n_estimators': range(1, max_estimators, 1)}
        for criterion in ["entropy", "gini"]:
            forest_reg = RandomForestClassifier(max_features=None,
criterion=criterion, random_state=119020)
            grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                                    scoring='roc_auc', n_jobs=3,
return_train_score=True)
            grid_search.fit(self.training_data, self.training_label)
            print(criterion, grid_search.best_params_)
            self.score_record[criterion] =
pd.DataFrame(grid_search.cv_results_)["mean_test_score"]
            self.test(grid_search.best_params_["n_estimators"], None,
criterion)

    def train_maxDepth(self, max_depth=30):
        depthLst = []
        for max_depth in range(1, max_depth, 1):
            self.test(74, None, "gini", max_depth)
            depthLst.append(self.auc)
        self.score_record["max_depth"] = depthLst

    def test(self, estimators, features, mycriterion="gini",
max_depth=None):
        self.classifier = RandomForestClassifier(n_estimators =
estimators, criterion = mycriterion,
            max_features=features, random_state = 119020,
max_depth=max_depth)
```

```
        self.classifier.fit(self.training_data, self.training_label)
        self.max_depth = max_depth
        # Predicting the Test set results
        if self.train_test:
            print("*****Training Data Performance*****")
            self.testing_data = self.training_data
            self.testing_label = self.training_label
        y_pred = np.array([1]*self.testing_label.shape[0])
        # y_pred = self.classifier.predict(self.testing_data)
        y_pred1 = self.classifier.predict_proba(self.testing_data)
[:,1]
        y_pred[y_pred1<0.33] = 0
        self.auc = roc_auc_score(self.testing_label, y_pred1)
        self.estimators = estimators
        self.features = features
        target_names = ['class 1', 'class 2']
        result = classification_report(np.array(self.testing_label),
y_pred, target_names=target_names)
        if self.report:
            print("AUC for testing Data is: ", self.auc)
            print(result)
        if self.train_test:
            self.dataLoad()
            self.train_test = False

    def visualize_criterion(self):
        plt.figure(figsize=(12, 8))
        plt.plot(pd.DataFrame(self.score_record))
        plt.legend(["Criterion: Entropy","Criterion: Entropy"],
fontsize=14)
        plt.xlabel("Number of features",  font = {'family': 'normal',
'weight': 'bold', 'size': 14})
        plt.ylabel("Average AUC in 5-fold validation", font =
{'family' : 'normal', 'weight' : 'bold', 'size': 14})

    def visualize_maxDepth(self):
        plt.figure(figsize=(12, 8))
        plt.plot(pd.DataFrame(self.score_record))
        plt.legend(["Different AUC when n_estimator=74,
criterion=Gini"], fontsize=14)
        plt.xlabel("Number of max_depth",  font = {'family': 'normal',
'weight': 'bold', 'size': 14})
        plt.title("Choose Hyperparameter for Random Forest:
max_depth", font = {'family' : 'normal', 'weight' : 'bold', 'size':
18})
        plt.ylabel("AUC of Testing Data", font = {'family' : 'normal',
'weight' : 'bold', 'size': 14})

    def visualize(self):
        plt.figure(figsize=(12, 8))
        plt.plot(pd.DataFrame(self.score_record))
        plt.legend(["max_feature: None","max_feature:
Sqrt","max_feature: Log2"], fontsize=14)
        plt.xlabel("Number of features",  font = {'family': 'normal',
'weight': 'bold', 'size': 14})
        plt.ylabel("Average AUC in 5-fold validation", font =
{'family' : 'normal', 'weight' : 'bold', 'size': 14})
        plt.show()
        self.score_record = {}

    def visualize_feature(self):
        plt.figure(figsize=(12, 8))
        plt.plot(pd.DataFrame(self.score_record))
        plt.legend(["Criterion: Entropy","Criterion: Gini"],
fontsize=14)
        plt.xlabel("Number of Maximum Decision Trees in the Model",
 font = {'family': 'normal', 'weight': 'bold', 'size': 14})
        plt.title("Choose Hyperparameter for Random Forest:
Criterion", font = {'family' : 'normal', 'weight' : 'bold', 'size':
18})
        plt.ylabel("AUC in testing Data", font = {'family' : 'normal',
'weight' : 'bold', 'size': 14})
        plt.show()
        self.score_record = {}
    def tree_visualize(self):
        important_attr = pd.DataFrame([self.training_data.columns, 
self.classifier.feature_importances_]).T.sort_values(by=[1], 
ascending=False).head(10)
        sns.set_theme(style="darkgrid",font="Microsoft 
YaHei",palette=sns.color_palette("tab10"))
        plt.figure(figsize=(12,8))
        ax = sns.barplot(x=0, y=1, data=important_attr)
        ax.set_title("Top 10 Important Attributes in Random 
Forest\nn_estimator: {}, max_feature: {}, max_depth: 
{}".format(self.estimators,self.features, self.max_depth), \
            fontdict={"family": "normal", "weight": "bold", "size": 
16})
        ax.set_xticklabels(ax.get_xticklabels(),rotation = 30, 
fontdict={"family": "normal", "weight": "bold"})
        ax.set_xlabel("Attribute Name", fontdict={"family": "normal", 
"weight": "bold", "size": 14})
        ax.set_ylabel("Importance", fontdict={"family": "normal", 
"weight": "bold", "size": 14})

    def draw_roc(self):
        score = self.classifier.predict_proba(self.testing_data)[:,1]
        auc = roc_auc_score(self.testing_label, score)
        fpr, tpr, _ = roc_curve(self.testing_label, score)
        plt.figure(figsize=(12, 6))
        plt.subplot(121)
        plt.plot(fpr, tpr, color='darkorange',
                label='ROC curve (area = %0.2f)' % auc)
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for testing set')
        plt.legend(loc="lower right")
        score = self.classifier.predict_proba(self.training_data)[:,1]
        auc = roc_auc_score(self.training_label, score)
        fpr, tpr, _ = roc_curve(self.training_label, score)
        plt.subplot(122)
        plt.plot(fpr, tpr, color='darkorange',
                label='ROC curve (area = %0.2f)' % auc)
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for training set')
        plt.legend(loc="lower right")
        plt.show()


if __name__=="__main__":
    p=Randomforest()
    p.train(10)
    p.visualize()
    p.train_criterion(10)
    p.visualize_feature()
    p.train_maxDepth(10)
    p.visualize_maxDepth()
    p.report=True
    p.train_test = False
    p.test(estimators=74, features=None, mycriterion="gini",
max_depth=9)
    p.tree_visualize()
    p.draw_roc()
```

7.2.9 Code for Neural Network

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.utils import shuffle
# for modeling
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping,ReduceLROnPlateau
# set random seed
my_seed = 80
np.random.seed(my_seed)
import random
random.seed(my_seed)
tf.random.set_seed(my_seed)
# package for checking results
from sklearn.metrics import
classification_report,confusion_matrix,roc_auc_score, roc_curve

class nnClassifier():
    def __init__(self, train_address = '../data/pca_train.csv',
test_address = '../data/pca_test.csv'):
        self.lam = None
        self.lr = None
        self.train_address = train_address
        self.test_address = test_address
        self.dataLoad()

    def dataLoad(self):
        self.origin_train = pd.read_csv(self.train_address)
        self.origin_train = shuffle(self.origin_train)
        self.origin_test = pd.read_csv(self.test_address)

    def nnFitting(self,N_HIDDEN = 80, activate = "tanh", optimizer =
tf.keras.optimizers.Adamax(learning_rate=0.001)):
        '''
        input: hidden units, activation function and optimizer
                 choose activation function from [relu, sigmoid,
softplus, softmax, softsign, tanh, selu, elu]
```

```
                 choose optimizers from [Adam, Nadam, Adagrad, Adamax,
Ftrl]
        output: the fitted model
        You can adjust the layers and units by yourself
        '''
        y = self.origin_train['class']
        x = self.origin_train.drop(['class'], axis = 1)
        # x_norm = x.apply(lambda t:(t - np.min(t)) / (np.max(t) -
np.min(t)))
        # x_norm = x.apply(lambda t:((t - np.mean(t)) / np.std(t)))
        x_norm = x
        X = np.array(x_norm)
        self.model = Sequential()
        # choose activation function from [relu, sigmoid, softplus,
softmax, softsign, tanh, selu, elu]

        activate = activate

        # add layers in the nn model
        self.model.add(Dense(N_HIDDEN, input_shape=(X.shape[1],),
activation=activate)) # Add an input shape (features,)
        # model.add(Dropout(.3))
        self.model.add(Dense(N_HIDDEN, activation=activate))
        # model.add(Dropout(.3))
        self.model.add(Dense(N_HIDDEN, activation=activate))
        # model.add(Dropout(.3))
        self.model.add(Dense(N_HIDDEN, activation=activate))
        # model.add(Dropout(.3))
        self.model.add(Dense(1, activation='sigmoid'))
        self.model.summary()

        # compile the model
        # choose optimizers from [Adam, Nadam, Adagrad, Adamax, Ftrl]
        opt = optimizer
        self.model.compile(optimizer = opt,
                    loss='binary_crossentropy',
                    metrics=['accuracy'])
        # callbacks
        es_val_loss = EarlyStopping(monitor='val_loss',patience = 2)
        reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
                                        patience=2, verbose=1,
mode='auto',
                                        min_delta=0.0001, cooldown=0,
min_lr=1e-8)

        # for unblanced dataset, set weights for class_i by (1 /
number of class_i)
        counts = np.bincount(y)
        weight_for_0 = 1.0 / counts[0]
        weight_for_1 = 1.0 / counts[1]
        class_weight = {0: weight_for_0, 1: weight_for_1}

        # now we just update our model fit call
        self.history = self.model.fit(X,
                                y,
                                callbacks=[reduce_lr, es_val_loss],
                                epochs=100, # you can set this to a
big number
                                batch_size=128,
                                validation_split = 0.25,
                                shuffle=True,
                                class_weight = class_weight,
                                verbose=1)

    def nnFittingVisualization(self):
        '''
        give the result on training and validation set
        output: two graphs, one for loss and one for accuracy
        '''
        history_dict = self.history.history
        # Learning curve(Loss)
        # let's see the training and validation loss by epoch
        # loss
        loss_values = history_dict['loss'] # you can change this
        val_loss_values = history_dict['val_loss'] # you can also
change this

        # range of X (no. of epochs)
        epochs = range(1, len(loss_values) + 1)

        # plot
        plt.plot(epochs, loss_values, 'bo', label='Training loss')
        plt.plot(epochs, val_loss_values, 'orange', label='Validation
loss')
        plt.title('Training and validation loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()

        acc = self.history.history['accuracy']
        val_acc = self.history.history['val_accuracy']

        # range of X (no. of epochs)
        epochs = range(1, len(acc) + 1)

        # plot
        # "bo" is for "blue dot"
        plt.plot(epochs, acc, 'bo', label='Training accuracy')
        # orange is for "orange"
        plt.plot(epochs, val_acc, 'orange', label='Validation
accuracy')
        plt.title('Training and validation accuracy')
        plt.xlabel('Epochs')
        plt.ylabel('Accuracy')
        plt.legend()
        plt.show()

    def testResult(self):
        '''
        give the results on test set
        '''
        test_y = self.origin_test['class']
        test_x = self.origin_test.drop(['class'], axis = 1)
        # you can choose to standard the dataset if you haven't done
it in the preprocession
        # test_norm = test_x.apply(lambda t:(t - np.min(t)) /
(np.max(t) - np.min(t)))
        # test_norm = test_x.apply(lambda t:((t - np.mean(t)) /
np.std(t)))
        test_norm = test_x
        test_x = np.array(test_norm)
        predicition = self.model.predict(test_x)
        preds = np.round(self.model.predict(test_x),0)
        confu_matrix = confusion_matrix(test_y, preds)
        result_report = classification_report(test_y, preds)
        AUC = roc_auc_score(test_y, predicition)
        print(confu_matrix)
        print(result_report)
        print(AUC)

        score = self.model.predict(test_x)
        auc = roc_auc_score(test_y, score)
        fpr, tpr, _ = roc_curve(test_y, score)
        plt.figure(figsize=(12, 6))
        plt.subplot(121)
        plt.plot(fpr, tpr, color='darkorange',
                label='ROC curve (area = %0.2f)' % auc)
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for testing set')
        plt.legend(loc="lower right")
        train_x = self.origin_train.drop(['class'], axis = 1)
        train_y = self.origin_train['class']
        score = self.model.predict(train_x)
        auc = roc_auc_score(train_y, score)
        fpr, tpr, _ = roc_curve(train_y, score)
        plt.subplot(122)
        plt.plot(fpr, tpr, color='darkorange',
                label='ROC curve (area = %0.2f)' % auc)
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC for training set')
        plt.legend(loc="lower right")
        plt.show()
        return preds

if __name__=="__main__":
    p = nnClassifier()
    p.nnFitting()
    p.nnFittingVisualization()
    p.testResult()
```

7.3 Code of Preprocessing

7.3.1 Exploring Data Analysis

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt



#%% Read data
train = pd.read_csv("..\\bankruptcy data\\train.csv")
train_X = train.iloc[:,:-1]
train_y = train.iloc[:,-1]
```

```
test = pd.read_csv("..\\bankruptcy data\\test.csv")
test_X = test.iloc[:,:-1]
test_y = test.iloc[:,-1]

#%% distribution of feature
plt.subplot(121)
plt.pie([train_y[train_y==0].shape[0],train_y[train_y==1].shape[0]],la
bels=[0,1],autopct='%3.2f%%',textprops={'fontsize':18,'color':'k'})
plt.title("Train Set")
plt.subplot(122)
plt.pie([test_y[test_y==0].shape[0],test_y[test_y==1].shape[0]],labels
=[0,1],autopct='%3.2f%%',textprops={'fontsize':18,'color':'k'})
plt.title("Test Set")



#%% heatmap
train_X.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8)

corr = train_X.corr() # We already examined SalePrice correlations
plt.figure(figsize=(12, 10))

sns.heatmap(corr[(corr >= 0.7) | (corr <= -0.7)],
            cmap='viridis', vmax=1.0, vmin=-1.0, linewidths=0.1,
            annot=True, annot_kws={"size": 8}, square=True)
```

7.3.2 PCA

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 17 18:01:05 2022

@author: L
"""


import pandas as pd
from imblearn.over_sampling import BorderlineSMOTE ,ADASYN, SVMSMOTE,
SMOTE
from imblearn.over_sampling import RandomOverSampler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt


train = pd.read_csv("..\\bankruptcy data\\adj_imp_train.csv")
train_X = train.iloc[:,:-1]/10
train_y = train.iloc[:,-1]

test = pd.read_csv("..\\bankruptcy data\\adj_imp_test.csv")
test_X = test.iloc[:,:-1]/10
test_y = test.iloc[:,-1]


#%%

smo = BorderlineSMOTE(random_state=0)
# smo = ADASYN(random_state=0)
# smo = SVMSMOTE(random_state=0)
# smo = SMOTE(random_state=0)
# ros = RandomOverSampler(random_state=0)
# ros.fit(train_X, train_y)
# X_resampled, y_resampled = ros.fit_resample(train_X, train_y)

X_resampled, y_resampled = smo.fit_resample(train_X, train_y)

new = X_resampled.join(y_resampled)

new.to_csv("..\\bankruptcy data\\smote_train.csv",index=False)

#%%
pca = PCA(0.97)
pca.fit(train_X)

var = pca.explained_variance_ratio_[0:10] #percentage of variance
explainedlabels =
['PC1','PC2','PC3','PC4','PC5','PC6','PC7','PC8','PC9','PC10']
plt.figure(figsize=(15,7))
plt.bar(['PC1','PC2','PC3','PC4','PC5','PC6','PC7','PC8','PC9','PC10']
,var)
plt.xlabel('Pricipal Component')
plt.ylabel('Proportion of Variance Explained')

newX = pca.transform(train_X)
new_testX = pca.transform(test_X)

newX = pd.DataFrame(newX)
new = newX.join(train_y)

new_testX = pd.DataFrame(new_testX)
new_test = new_testX.join(test_y)

# new.to_csv("..\\bankruptcy data\\pca_train.csv",index=False)
# new_test.to_csv("..\\bankruptcy data\\pca_test.csv",index=False)
```

7.3.3 Handling Missing Value

```
train = read.csv('../bankruptcy data/train.csv')
test = read.csv('../bankruptcy data/test.csv')
train_X = train[-65]
test_X = test[-65]
train_Y = train[65]
test_Y = test[65]
head(train)
head(test)
summary(train)
library(mice)

library(dplyr)
train.miss = train %>%
  select_if(~any(is.na(.)))
test.miss = test %>%
  select_if(~any(is.na(.)))

md.pattern(train.miss,rotate.names = T)
```

```r
library(VIM)
mice_plot <- aggr(train.miss, col=c('navyblue','yellow'),
                   numbers=TRUE, sortVars=TRUE,
                   labels=names(train.miss), cex.axis=.7,
                   gap=3, ylab=c("Missing data","Pattern"))

imputed_train <- mice(train_X, m=5, maxit = 5, seed = 500)

stripplot(imputed_train, Attr37 + Attr27 + Attr45 ~
 .imp,col=c("grey",mdc(2)),pch=c(1,20))

xyplot(imputed_train , Attr45 ~  Attr37| .imp, pch=20,cex=1.2)

densityplot(imputed_train,data = ~Attr37 + Attr27 + Attr45)


completed.Train <- complete(imputed_train,2)

library(caret)
library(RANN)
miss = preProcess(completed.Train,method='bagImpute',k=5)
imputed.train = predict(miss,completed.Train)
imputed.train["class"] = train_Y

total = rbind(imputed.train,test)
total_X = total[-65]
total_Y = total[65]
imputed_total <- mice(total_X, m=5, maxit = 5, seed = 500)
completed.Total <- complete(imputed_total,2)

library(caret)
library(RANN)
miss = preProcess(completed.Total,method='bagImpute',k=5)
imputed.total = predict(miss,completed.Total)
imputed.total["class"] = total_Y
imputed.test = imputed.total[950:1100,]
```