ML Final Project

鍾孟辰

b09508011@ntu.edu.tw
National Taiwan University
Taipei, Taiwan

1 Introduction

Kaggle, a data modeling and data analysis competition platform, holds competition of data science for statisticians' and data mining experts. Participants experimented with different methods, competing against each other for the prize to the best model. [1]

Though Kaggle competitions are incredibly fun and rewarding, they are intimidating for novice like me in data science. [2] Therefore, instead of participating in a Kaggle competition and lose my enthusiasm in machine learning, I joined a playground competition. Tabular Playground Series organizes a playground competition once a month, for my final project, I choose to participant in the one held in March 2021.

The submissions of the competition are evaluated on area under the ROC curve between the predicted probability and the observed target. There will be 30,000 sets of data for training and 20,000 sets for testing. For each set of data, there are 30 kinds of features, including bool, float, and string.

Since there is no limitation on choosing model in the competition, I tried models I've learnt in this semester, including clustering, linear discriminate, and artificial neural network. As for the reason why I didn't try nonparametric methods was that after trying clustering, I find it hard to train models efficiently by letting data speak for themselves since the dataset is massive.

2 Motivation

Being a student studying biomedical engineering, I considered it is important for me to learn machine learning well on technical level and medical level, that is, I should be able to write a good model while be aware of what kind of clinical problems might happen. Before learning machine learning from the aspect of biomedical engineering, I would like to approve on my technical level. In result, I took part in playground competition in Kaggle as my final project, hopping I could learn machine learning better by training a synthetic, beginner-friendly dataset (but based on a real dataset and generated using a CTGAN).

3 Try and Errors

3.1 Preprocessing

In this dataset, the features are composed of different data types, including bool, float, and string (they are categorical).

However, we usually prefer the data type of the features be numeric. Therefore, I tried two methods, respectively to be label encoder and one hot. The former target labels with value between 0 and n_classes – 1, transforms non-numerical labels (as long as they are hashable and comparable) to numerical labels. [3] The latter creates a binary column for each category and returns a sparse matrix. [4] After transforming all the non-numerical features into numerical features, I used linear regression to test which methods is better. The results show that using one hot could get a lower loss and higher accuracy. Thus, I choose one hot to transform the non-numerical features. After transforming the number of features increased from 30 to 634. Then, I randomly split the dataset into 240,000 training data and 60,000 validation data, and combine every 30,000 data into a loader.

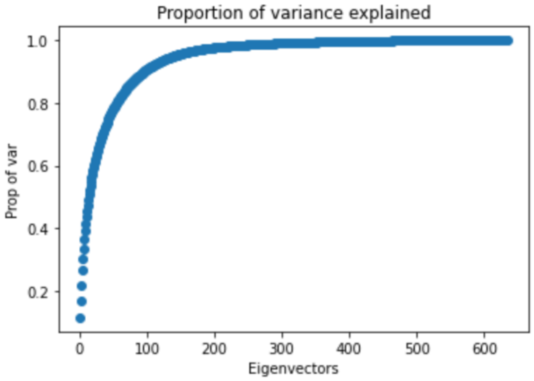
Metrics	Training	Validation
Number of data	240,000	60,000
Loader size	8	2

Table 1: Data Distribution

3.2 Dimension reduction

To clarify whether using feature selection or feature extraction is better, I plot the histogram of the distribution of each feature and find out that they all look similar. Since they all seem having similar distribution, I assumed that each feature attribute almost equally to the model. Therefore, I choose the feature extraction.

At first, I tried PCA and choose to use 100 components because the elbow of proportion of variance explained approximately happens at 100. After project data on the eigenvectors I choose, I use linear regression to check if I need to select more components or less. However, no matter how I change the numbers of selected components, the result doesn't change much. Later, it occurred to me that maybe I do not necessarily need to do dimension reduction since the ratio of features number and data is 634/240,000, equals to 0.0026, which is lesser than 1%. Therefore, I decided not to do dimension reduction, unless overfitting happens.



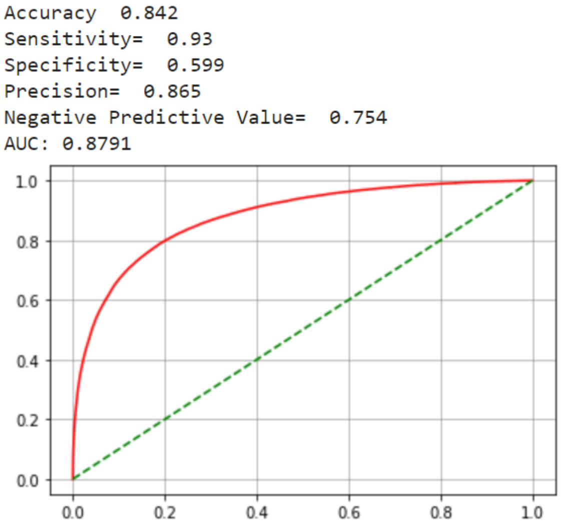
Graph 1: Proportion of variance explained

3.3 Selecting Model

Linear discrimination: I tried linear regression and logistic regression.

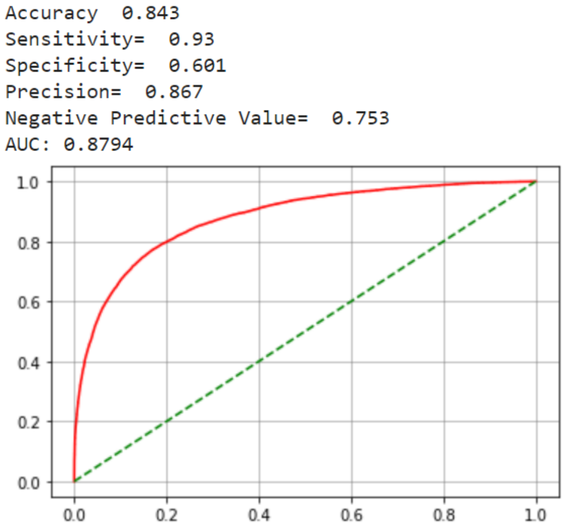
For linear regression, I simply solve the linear equation without using loss function or optimizer to adjust the model. The training area under ROC curve (simply auc) is 0.8814, while the validation auc is 0.8799. With this result I was having confident that the auc would at least reach to 0.9 if loss function and gradient decent is applied.

For logistic regression, I used the module in sklearn to build model. The result was fine, too. The training auc is 0.8791 and the validation auc is 0.8794.



```
Accuracy  0.842
Sensitivity=  0.93
Specificity=  0.599
Precision=  0.865
Negative Predictive Value=  0.754
AUC: 0.8791
```

Graph 2: performance of training data using logistic regression



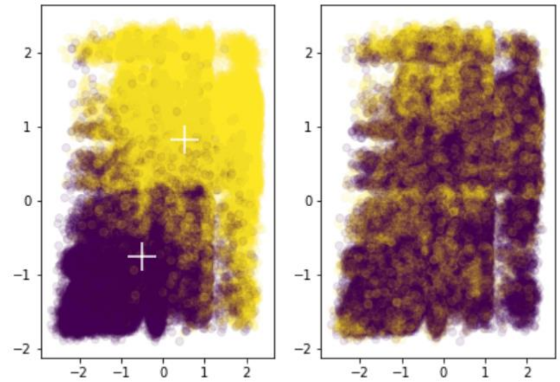
```
Accuracy  0.843
Sensitivity=  0.93
Specificity=  0.601
Precision=  0.867
Negative Predictive Value=  0.753
AUC: 0.8794
```

Graph 3: performance of validation data using logistic regression

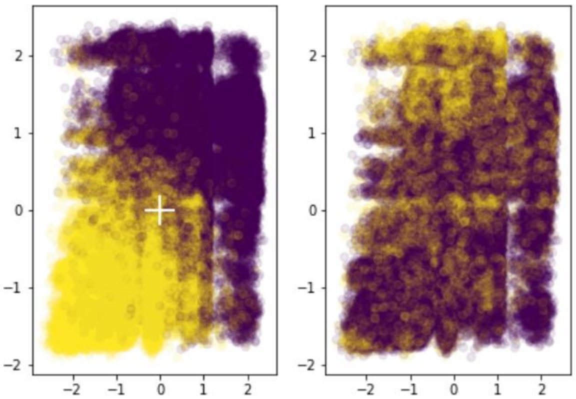
Clustering: I tried K-means, fuzzy C-Means, and Spectral clustering.

For K-means, I used the module in sklearn to build model. The accuracy for training data ranges from 0.5 to 0.8. I assumed that it is because the first two center is chosen randomly, thus the accuracy flows.

For fuzzy C-mean, I used the module in fcmeans to build the model. The accuracy for training data ranges from 0 to 0.5. To figure out what happened, I plot the scatter plot to see the distribution of the data.

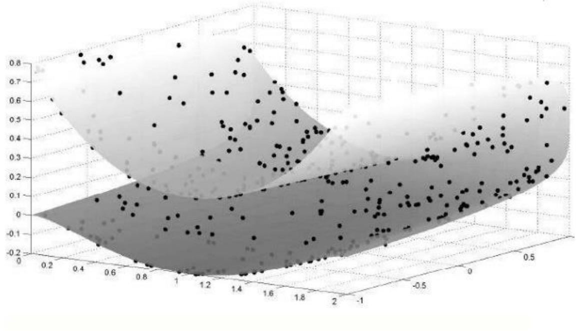


Graph 4, 5: K-means centers and data scatter plot with prediction of k-means (4: left) and data scatter plot with ground truth (5: right)



Graph 6, 7: fuzzy C-means data scatter plot with prediction of fuzzy C-means (6: left) and data scatter plot with ground truth (7: right)

The graph 4 and graph 6 change in every run. In graph 5 (same as 7), the data with different label mix together, therefore, I conclude that this dataset is not suitable for clustering. However, the plot reminded me of the graph down below:



Picture 1: the 3-D from slide clustering_v2

I thought, maybe the data seems close in 2-D but actually far in 3-D. Therefore, I wanted to try spectral clustering. I used module in sklean to build model. However, it occupied too many RAM resources on Colab and was forced to shut down every time I tried to run the code. I tried to consume the number of features and data, but it still wouldn't work. Thus, I gave up on this option.

Moreover, after seeing graph 5, I assume that non-parametric methods such as K-NN wouldn't work well as well. In addition, there are 300,000 data, which would take lots of RAM resources to run non-parametric model since every data needs to speak for themselves. Therefore, I decided not to try non-parametric method.

After I get the conclusion above, it occurred to me that the predictions of the clustering are not probabilities. Therefore, auc couldn't be evaluated.

Artificial Neural Network: I tried DNN.

I took the code from homework 3 and tried 2 hidden layers, 3 hidden layers, different nodes number, and different learning rate. However, the accuracy couldn't be higher than 60% no matter how hard I tried.

Comparison: Among all kinds of model, logistic regression seems to have the best performance. To tune the parameters, I decided to build a model myself instead of using module.

4 Experiments

4.1 Building model

Before building a logistic regression model, I added an all-ones column as feature, which act as the constant in the linear problem. In formulas down below, y is ground truth, x is dataset, and w is weight in the linear model.

Prediction: The function for prediction is the transpose matrix of w dot transpose matrix of x plus the constant w^0, and then put the result into sigmoid function.

$$prediction\ y^t = sigmoid(w^T x^T + w^0)$$

Loss function: I tried Lasso function, MSE function, and cross entropy. After testing three kinds of function with validation data, cross entropy had the best performance, thus, the model built with sklearn use cross entropy, too. Therefore, I choose cross entropy as my loss function.

$$loss = \frac{1}{N}\sum_{n=1}^{N} ln(1 + exp(-y_n w^T x_n))$$

Gradient: I do the gradient decent by trying to minimize the gradient written down below. The weight matrix will then minus learning rate × gradient to achieve gradient decent.

$$gradient = \frac{1}{N}\sum_{n=1}^{N} \partial(-y_n w^T x_n)(-y_n x_n)$$

4.2 Experiment 1

I tested different learning rate with 0.01, 0.03, 0.05, 0.07, for each I set the epoch to be 50, and the result is shown in Table 2. As Table 2 shows, the lower the learning rate is, the more gently the curves are, and eventually converge to a horizontal line.

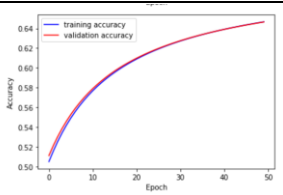
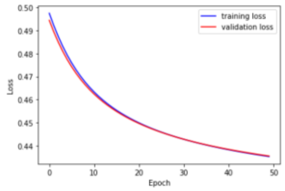
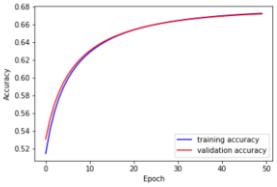
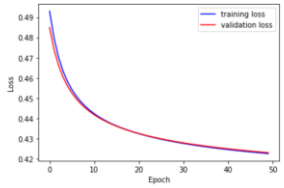
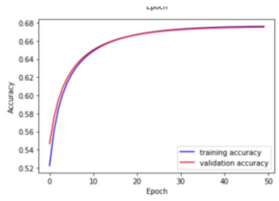
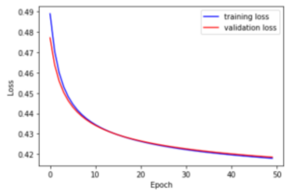
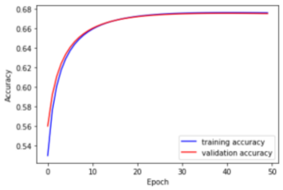
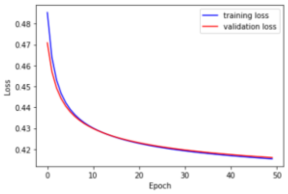
	Accuracy	Loss
0.01		
0.03		
0.05		
0.07		

Table 2: performances for different learning rates

The accuracy for different learning rate with 0.01, 0.03, 0.05, 0.07 is 65%, 67%, 68%, 68% (for both training and validation data) respectively. I would choose 0.05 as final learning rate since the slope of the curve looks small enough and won't reach the highest too fast. However, the slope of loss curve for neither 0.05 nor 0.07 aren't small enough. Therefore, I would like to try running a bigger epoch.

4.3 Experiment 2

After the experiment 1, the learning rate was fixed, and the next step is to see if the loss could be lower, and the result in shown in Table 3.

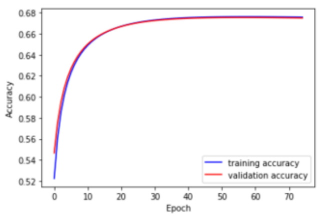
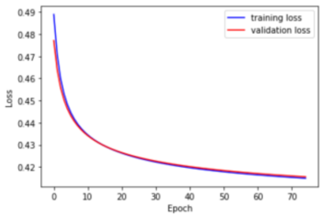
Accuracy	Loss
	

Table 3: performances when there are 75 epochs

As the result shown in graph 3, the slope of loss curve approaches to zero, and the accuracy remain the same.

5 Result

5.1 Dataset

Since the features in the dataset need to be numerical, I used one hot method to transform non-numerical features into numerical. However, after transforming, the feature number for training dataset is not equal to the one for testing dataset. The testing dataset is four feature lesser, thus, I added four all-ones columns as constant in logistic regression to make sure the size of training dataset equals to testing dataset.

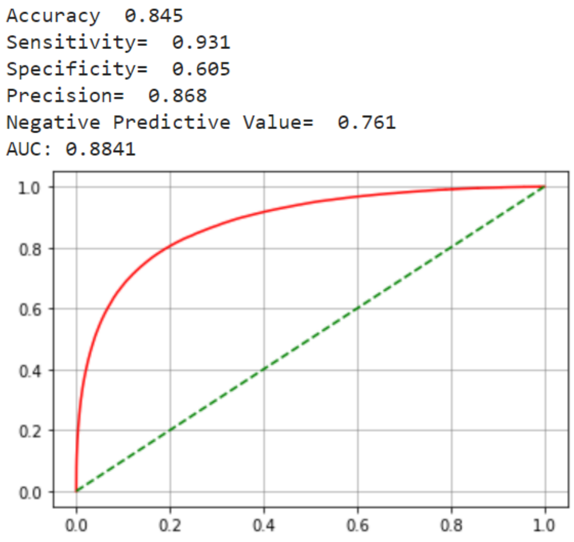
	Train	Test
Number of data	300,000	200,000
Number of features	634	634

5.2 Training Model

In experiment, it is obvious to see that the logistic regression I built myself doesn’t work better than the one in sklearn no matter in result or time complexity, so I use the module to get the result for the competition. The penalty used in the model is l2 [5], and since the module’s default is to add constant automatically, I didn’t add an extra column.

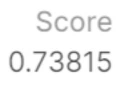
5.3 Performance

The auc of testing dataset is 0.8841, and the performance is shown down below.



```
Accuracy  0.845
Sensitivity=  0.931
Specificity=  0.605
Precision=  0.868
Negative Predictive Value=  0.761
AUC: 0.8841
```

The score of the testing data is 0.73815.



Score
0.73815

6 Conclusion

7 Discussion and Future Works

8 Reference