Don’t overfit competition

Liu Yuzhou(21100602d)

Responsable of the experiment or other information

1. Introduction

This report demonstrates the implementation for the Kaggle competition: Don’t Overfit II. In the following part, I will first explain the dataset, then demonstrate how my model was built and how it’s gonna be improved. Finally, I will summarize my work, and show my final solution.

2. Dataset

Before building my model, first I will try to analyse the attributes, and have an overview of the data set.

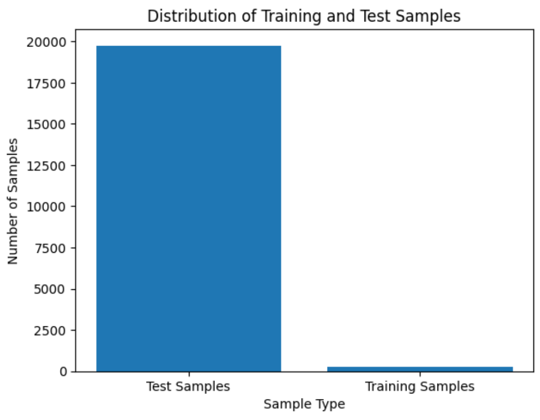


Figure 1. *Portion of training and testing set*

We can see that we only have 250 samples for training, but for the testing set, we have 19750 samples for testing. Because the number of training samples is too limited, We will easily encounter the problem of overfitting.

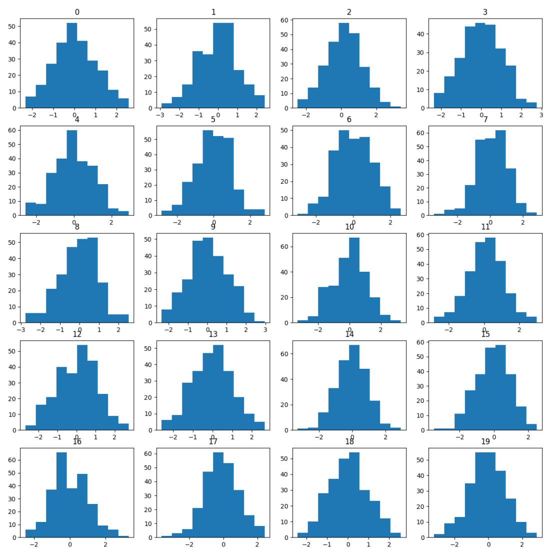


Figure 2. *View of first 20 columns*

We show the distribution of the first 20 attributes in **training set**, and we also checked the minimum standard among all attributes is 0.44, and the maximum is 1.12, and it doesn’t include any null value. The target is binary, which consists of 36% 0s and 64% 1s.

3. Method(Build model)

3.1. Overall

Before we start, we could think that as the training dataset is really small, by using any model, we can easily fit all the training samples but obtain a bad result on the testing set. So we need to consider normalising the dataset, removing outliers, or using a simpler model..., I think good performance doesn’t only rely on a good model, but also need to put more effort into getting better feature engineering.

3.2. Model selection

Because complex models have too many parameters, they can easily cause the problem of overfitting, so I decided to try some simple models. The following method is the result of the 10-fold validation result of each model(K-fold validation is really important here; e.g for SVM, I obtained a result of 0.8 when we randomly split the dataset, but after I used 10-fold to validate it, the average accuracy was only 70.3%).

Model/Classifier	Accuracy
SVM-Linear	70.8%
Random Forest Classifier	73.2%
KNN	69.6%
ExtraTrees classifier	73.2%
Logistic regression	74.8 %

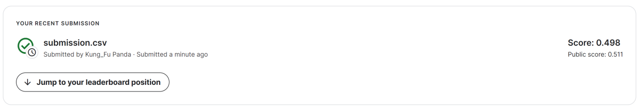
Table 1. *Training performance on validation set*

We can see that all the models give similar accuracy, so we cannot easily tell which is the best.

3.3. Two innovative Voting classifier

3.3.1. Model explanation.

As we can see from the model selection part, we can hardly select the best model among three different models. So here, I decided to build a voting classifier for them. Firstly, I try to regularisation the dataset using l1 and l2, respectively. The performance on the validation set is quite ok, which is about 70% accuracy. But the performance for the test set is pretty low, which is only about 50%, not better than a random guess.



YOUR RECENT SUBMISSION

submission.csv
Submitted by Kung_Fu Panda · Submitted a minute ago

Score: 0.498
Public score: 0.511

Figure 3. *Poor result*

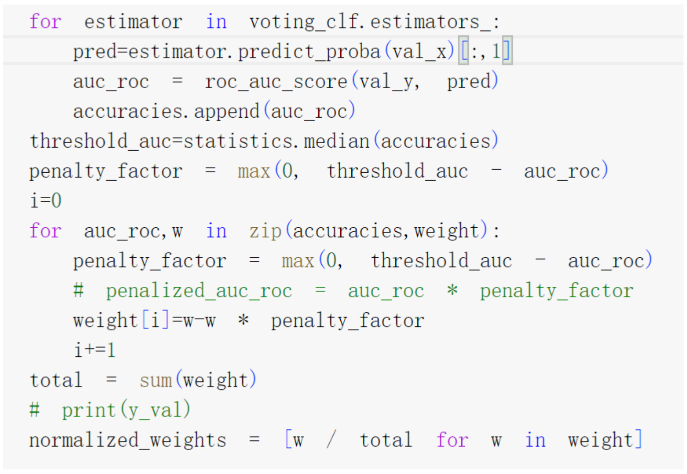
Potential reason: We did not consider the trustworthiness of prediction result for each model during training; we should

assign weights for the model.
I have tried two ways to improve the performance:
Way1: We have learned TP, TN, FP, and FN in lectures to measure the accuracy of our model. And AUC-ROC is a good way to summarise the accuracy from those perspectives. I decide to use AUC-ROC to measure the accuracy of our model. Based on the result I obtained, I will put a penalty on the model which worsens our performance. Following is the formula I used for putting penalties on weight, **threshold** is the median auc_roc value for the set of classifiers.

Updating weight:

$$penalty\_factor = max(0, threshold\_auc - auc\_roc)$$
$$weight = weight - weight * penalty\_factor$$

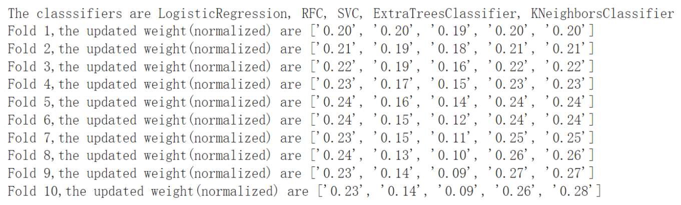
Following **Figure 4** shows the implementation of the updating weight strategy.



```
for  estimator  in  voting_clf.estimators_:
    pred=estimator.predict_proba(val_x)[:, 1]
    auc_roc  =  roc_auc_score(val_y,  pred)
    accuracies.append(auc_roc)
threshold_auc=statistics.median(accuracies)
penalty_factor  =  max(0,  threshold_auc  -  auc_roc)
i=0
for  auc_roc,w  in  zip(accuracies,weight):
    penalty_factor  =  max(0,  threshold_auc  -  auc_roc)
    #  penalized_auc_roc  =  auc_roc  *  penalty_factor
    weight[i]=w-w  *  penalty_factor
    i+=1
total  =  sum(weight)
#  print(y_val)
normalized_weights  =  [w  /  total  for  w  in  weight]
```

Figure 4. *Way for update weight*

Here, by taking advantage of K-fold validation, I can update the weight quite a few times. Instead of passing the whole dataset into it, I just do K-fold to the training set, which means I further split my training set into a sub-validation set and a sub-training set. The sub-training set is still used for training, while the sub-validation is not only used for validation. I used it to update the weight for different sub-models in the voting classifier.



```
The classsifiers are LogisticRegression, RFC, SVC, ExtraTreesClassifier, KNeighborsClassifier
Fold 1,the updated weight(normalized) are ['0.20', '0.20', '0.19', '0.20', '0.20']
Fold 2,the updated weight(normalized) are ['0.21', '0.19', '0.18', '0.21', '0.21']
Fold 3,the updated weight(normalized) are ['0.22', '0.19', '0.16', '0.22', '0.22']
Fold 4,the updated weight(normalized) are ['0.23', '0.17', '0.15', '0.23', '0.23']
Fold 5,the updated weight(normalized) are ['0.24', '0.16', '0.14', '0.24', '0.24']
Fold 6,the updated weight(normalized) are ['0.24', '0.15', '0.12', '0.24', '0.24']
Fold 7,the updated weight(normalized) are ['0.23', '0.15', '0.11', '0.25', '0.25']
Fold 8,the updated weight(normalized) are ['0.24', '0.13', '0.10', '0.26', '0.26']
Fold 9,the updated weight(normalized) are ['0.23', '0.14', '0.09', '0.27', '0.27']
Fold 10,the updated weight(normalized) are ['0.23', '0.14', '0.09', '0.26', '0.28']
```

Figure 5. *K-fold for updating weight*

But the training result is quite weird!!! I could only find 0s on my prediction result. Then, I began to wonder the reason. After checking the AUC-ROC chart, I found that all the models performed poorly. All the models obtain AUC-ROC values around 0.63, and we can see the model has the trend to predict a label rather than another.

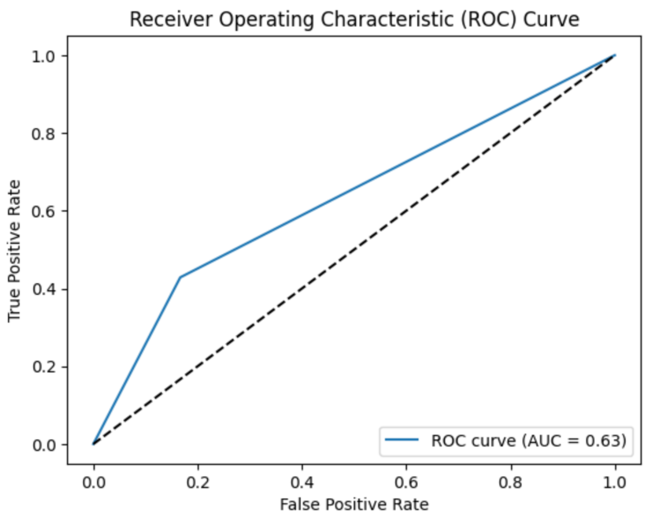
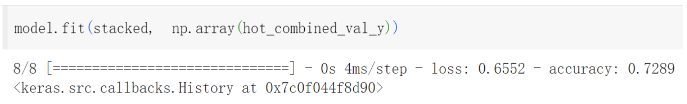


Figure 6. *K-fold for updating weight*

Way2: As we have already learned the MLP model in class, I am considering whether I can feed the prediction result of the five different models into the model, which means to have **five different input channels**, and I decide to build a simple model which only has three layers(one input layer, one hidden layer and a one channel output layer).
The way for training is similar to **way1**; we also use K-fold to split the training set, but here I concatenated all the results for validation result together, then fed them into the MLP classifier in one go.



```
model.fit(stacked,  np.array(hot_combined_val_y))
```

```
8/8 [==============================] - 0s 4ms/step - loss: 0.6552 - accuracy: 0.7289
<keras.src.callbacks.History at 0x7c0f044f8d90>
```

Figure 7. *MLP for combined "sub-validation set"*

Unfortunately, this way doesn’t work well, as the validation result is always poor. I have tried to change the optimiser and modify the layered architecture(e.g. convert the label into one hot encoded format, then apply the softmax function in the output layer). But none of them works. Perhaps SGD and Adam optimiser are not suitable here, or we need a more complex architecture(Considering there is also a risk of overfitting). I will focus on optimising the result for the **Way1** model.

3.3.2. Reflection.

1.(**Poor performance on MLP**) I was expecting that the MLP model would work well in this case. Because the large number of neurons are more likely to gain information from different perspectives, but the truth is that the performance is not good. I think the major problem is the amount of the dataset is too limited, so that the model could just memorise the dataset, without learning the relationship between different features.
2.(**SVM & RFC does not perform well**) I did a lot of research and tried to improve the performance of these two models. I think it is because the dimensionality of the data is too high; under such a situation, the performance of these two models is hard to improve. Hopefully, we can solve this by feature selection.

4. Attempt for improvement

How is that possible? After I checked the voting classifiers, I knew the reason: nearly all of them encountered the problem of overfitting, so without any good prediction for each sub-model, it is reasonable to get such an unsatisfied result then. I am thinking about the problem related to the dataset, I have tried regularisation the dataset, but haven't tried different lambda values. So can we use good feature engineering to predict each model better? Now lets try!!!

4.1. Balanced labels

The first thing I could see from the training dataset is that the majority of the samples are labelled 1, while only about 1/3 of the samples are labelled 0. That might cause the model to tend to predict the label as 1 in order to minimize the error(We definitely don't want it to learn such a thing).

Then I try think about how to make the label balanced. There are two ways to achieve that. **(1)**Simply randomly select and duplicate the samples labelled one in the train set. **(2)**Resample the train set with the same number of objects labelled 0 and 1 with the minority number.

Both methods have some **drawbacks** here; method (1) is not good for our way of training, as we want to further split the training set(Duplicate for weights update in "sub-validation set"). The drawback of method (2) is quite obvious, as we already have limited samples for training; if we discard some of the samples, we will lose a lot of information.

Then, after searching, I found a middle way which could help be helpful to overcome the situation. That is to use **SMOTE** to generate some synthetic samples based on the point and its K-nearest neighbours.

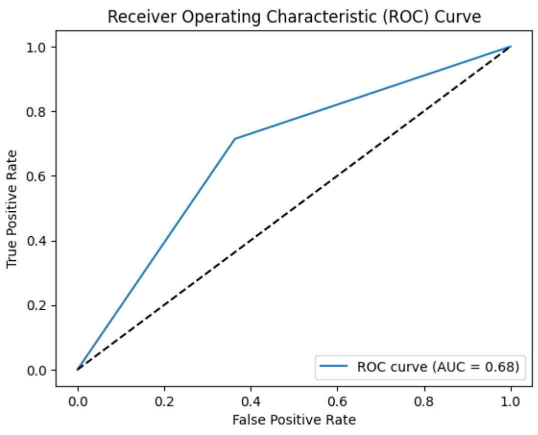


Figure 8. *AUC-ROC after synthetic sampling*

According to **figure 8**, the AUC-ROC curve for the validation set is much better. But still, though it is not that good, we can see that the AUC-ROC score of our model have improved a lot.

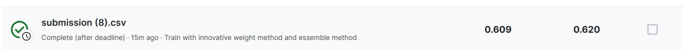

submission (8).csv
Complete (after deadline) · 15m ago · Train with innovative weight method and essemble method
0.609 0.620

Figure 9. *Improved result for synthetic samples*

Then I test the effect of resampling, **Figure 9** shows the improved result after I enlarged the dataset with some synthetic samples. We can see that the result improved a lot. Also, here I found something really nice: the score we obtained from the validation set each time is similar to our submission score, which means the validation set reflects the real scenario.

4.2. Scale the data

As the training data and test data have a large size difference, the scales of them is also quite different. We can see **Figure 8**, the mean and variance lie far away from each other.

```
print(test.drop("id",axis=1).values.mean().sum()/300)
print(test.drop("id",axis=1).values.std().sum()/300)
```

```
-6.396208157524615e-07
0.003334743652759724
```

```
print(dataset.mean().sum()/300)
print(dataset.std().sum()/300)
```

```
0.41651226666666674
1.2439330634431598
```

Figure 10. *Scale of the training and testing data*

So what about scaling them, in order to make the test and training datasets have a similar distribution? I believe this can also improve our performance. Afterwards, the training will be based on the scaled training and testing data.

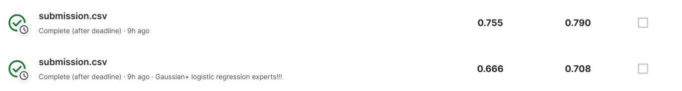
4.3. GaussianNB work as meta-classifier

4.3.1. Method explanation.

As **MLP** model does not work well, the weighting method may not be suitable as it may cause ignorance of some models(As their prediction accuracy is not that high, but the probability number may also contain some information). So, I am considering using the GaussianNB classifier as the final decision maker.

Or even a potentially better solution, I combine both methods together to attain a higher accuracy submission. Also, I could see from the weight we obtained that logistic regression tends to make a better prediction on unseen data; it may be caused by the simplicity of the model. So now I am also trying to change the experts in the voting classifier to different versions of logistic regression.

Now, we test the model's performance (consisting of 3 logistic experts, and the model hasn't been weighted). I found it quite hard to weigh the labels, because multiplying the weight by those labels doesn't make sense. It always generates the same result; I think the potential reason could be that the GaussianNB model is implemented based on the idea of naive Bayes, as the multiplier doesn't really make the distribution vary.


submission.csv
Complete (after deadline) · 9h ago
0.755 0.790

submission.csv
Complete (after deadline) · 9h ago · Gaussian+ logistic regression experts!!!
0.666 0.708

Figure 11. *GaussianNB and 3 logistic experts*

Different experts have different parameter settings. Some apply l1 regularisation, while others apply l2. It seems to enhance the prediction result from different perspectives. We can also discover from **Figure 11** that a better result can be obtained from predicting by probabilities. That might caused by the tied case, which can not be easily classified, but by incorporating elasticity into the prediction results, we can classify these cases better, and we can see the result boosted by nearly 0.1.

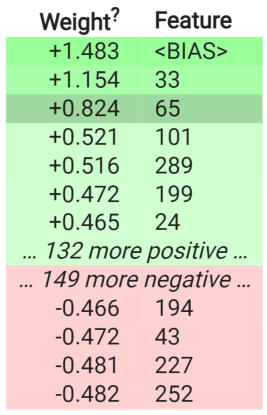
4.3.2. Reflection.

Although the GaussianNB classifier doesn't generate nice quality output here, it is suitable for making a nice decision based on the observation of the three logistic experts, as it is implemented based on the Naive Bayes method, which can take different weights on considering different attributes.

4.4. Feature selection

4.4.1. Method explanation.

High dimensional data is also a potential reason to get the model overfit, because there will likely be redundant information, which will worsen the result. What I want to try to improve the performance here is to reduce the dimensionality by reducing the number of attributes.



Weight?	Feature
+1.483	<BIAS>
+1.154	33
+0.824	65
+0.521	101
+0.516	289
+0.472	199
+0.465	24
… 132 more positive …	
… 149 more negative …	
-0.466	194
-0.472	43
-0.481	227
-0.482	252

Figure 12. *Feature importance for logistic(elit5)*

We can see from **Figure 12**, that different features have different impacts for logistic regression. Some may even have a negative impact on the predicted result. Then, I will try different methods to select features.

Model/classifier	Submission
Info-gain(AUC-ROC weight)	67.4%
Lasso(Gaussian decision maker)	80.0%
Lasso(AUC-ROC weight)	62.0%
Info-gain(Gaussian decision maker)	78.4%

Table 2. *Submission performance*

4.4.2. Reflection.

1. (**Info-gain(AUC-ROC weight)**) The information gain feature selection method seems to be appropriate for the innovative weight update method we mentioned earlier, though it does not beat the best result. But it boosts the original result of this model from 0.62 to 0.674, and it only selected 24 features for training, which means the training is also much faster.
2. (**Correlation overcome by scaling**) Correlation columns presented in the same dataset could also worsen the result, but here, after checking by function, I didn't obtain any correlation columns from the scaled dataset. That means the scaling method might reduce the correlation between columns.
3. (**Lasso(Gaussian decision maker)**) The performance of this model is similar to the one without feature selection. But the good news is that we learn things with fewer features.
4. (**Info-gain(Gaussian decision maker)** At the beginning, I thought information gain might not be a good method for this Gaussian decision maker because the accuracy is worsened to 72%. But by changing the threshold for the information gain to let more columns be selected. Finally, I achieved accuracy score of 78% on the public leader board, and 76% on the private leader board(Which is slightly better than the one obtained previously). That means the model was improved as it works better than before for unseen data.

I believe that we can achieve better results by having better parameters; there are a lot of methods for doing that, such as grid search. But here I mainly focus on improving my model.

4.5. Outlier elimination

Outlier is also a factor which could affect the performance of the model. The model may try hard to fit it, but it worsens the performance on the normal data points. Now, we will try to eliminate the outlier by using Z-scores. Because the Z-score is calculated based on attributes, so I will first do feature selection, then eliminate the outliers based on the feature retained.

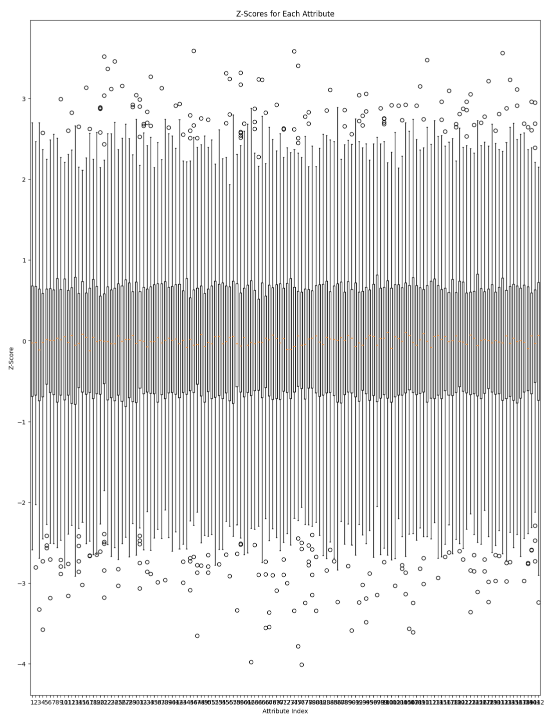


Figure 13. *Z-score result for info-gain*

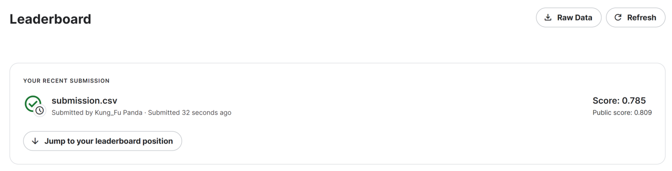
We can observe the result of the Z-score based on the result of info-gain feature selection in **Figure 13**; based on the plot chart, we can try different thresholds for outlier elimination. The following **Table 3** shows the result we have obtained.

Model/classifier	Submission
Info-gain(AUC-ROC weight)	72.7%
Lasso(Gaussian decision maker)	80.9%
Lasso(AUC-ROC weight)	80.6%
Info-gain(Gaussian decision maker)	74.8%

Table 3. *Submission performance after Z-score outlier elimination*

5. Final solution

The following **Figure 14** shows the best submission entry I have obtained.


Leaderboard

YOUR RECENT SUBMISSION

submission.csv
Submitted by Kung_Fu Panda · Submitted 32 seconds ago

Score: 0.785
Public score: 0.809

Figure 14. *Best submission entry*

The final solution has several steps. First, the training and test data are scaled separately. Then, feature selection is performed using the Lasso method. Outliers are identified and eliminated using the Z-score method. Next, three logistic experts are trained using K-fold cross-validation. The weights of each expert are updated based on their accuracy on the validation set, measured using the AUC-ROC method mentioned earlier. Finally, predictions are made using trained and weighted logistic experts.

6. SUMMARY

During the competition, I tried a lot of methods to overcome the problem of overfitting.

From the model's perspective, I developed a model that combines predictions from multiple experts using an ensemble method. And I tried three ways of decision-making**(Mentioned in Section 3)**. Finally, I found that the Gaussian meta-classifier and AUC-ROC weight updating method are useful, as they both achieve good accuracy. K-fold validation is an important technique as it improves the robustness of the model and enhances the training results.

From the data perspective, I tried to balance the label, scale the data, and select useful features; after the application of each technique, the results keep improving, which means all of them are useful.

Finally, I learned a lot from this experience. We not only need to have a good model in order to learn things effectively, but we also need to try our best to provide good feature engineering to our dataset in order to make the features of the dataset more representative(Making the knowledge much easier for the model to learn). So, by conducting research from both perspectives, we have achieved an accuracy of around 80%, which I think is quite decent and meets my expectations.