Don't Overfit! II: How to Prevent Overfitting?

ZHANG Caiqi 18085481d

The Hong Kong Polytechnic University

1 Introduction

In this report, we present the methods and results for the Kaggle competition: Don't Overfit II. After a detailed data exploration, we list the difficulties and challenges of this task. With respect to the most significant overfitting problem, we applied **seven** techniques to enhance the model's generalization ability. Finally, the model achieved 0.814 accuracy. We also conducted the ablation test to examine the effectiveness of each technique.

2 Data Exploration

First, we will explore some characteristics of the dataset to find some insights for the future prediction. According to the analysis, we can have the following findings:

- The test set (19750) is much larger than the training set (250).
- For each data point, it has 300 features without practical meaning.

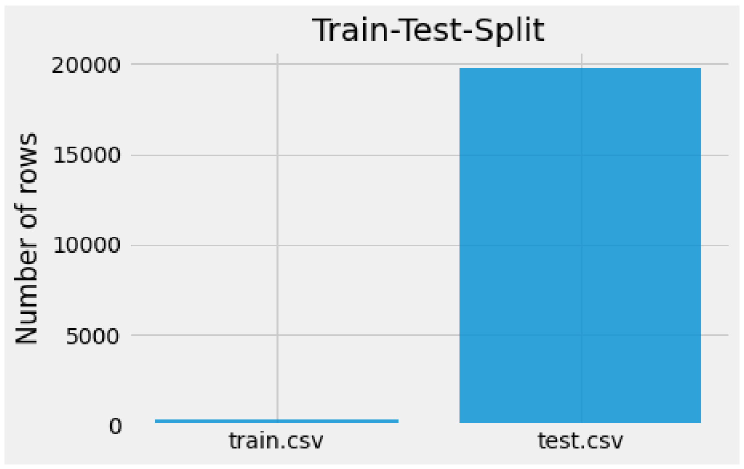


Figure 1: The number of data points in training set and test set

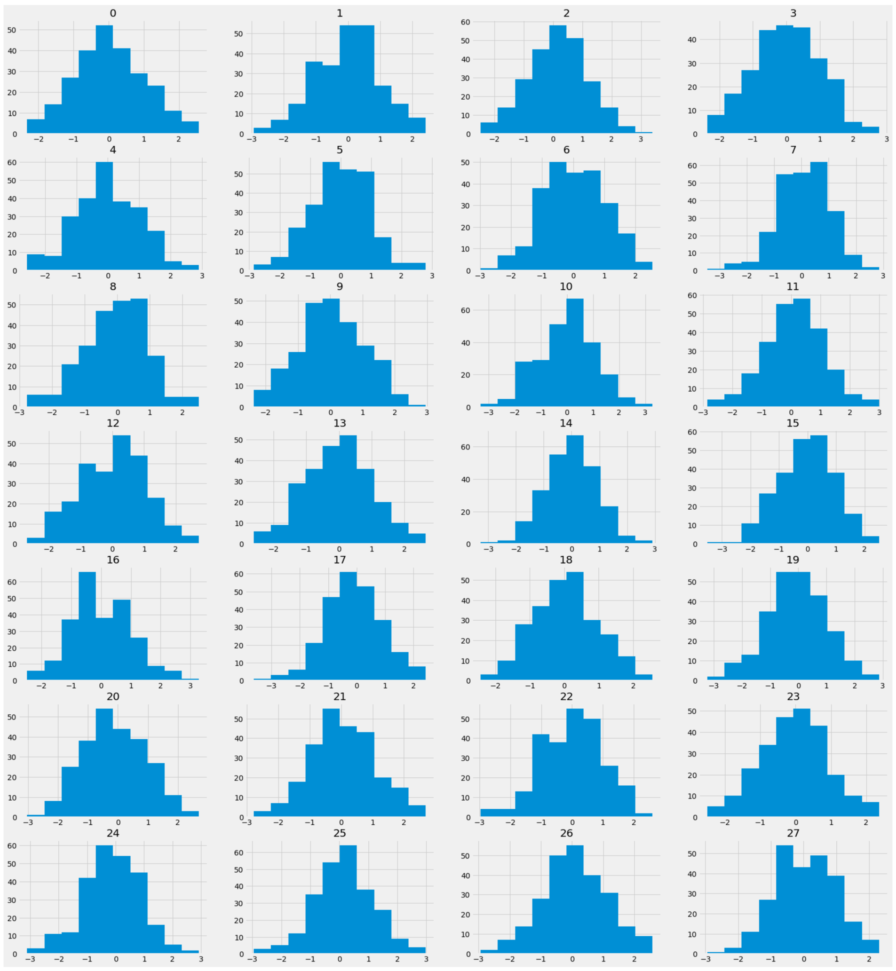


Figure 2: Distributions of first 28 columns

- The target is binary. 36% are 0 and 64% are 1.
- The standard deviation of the features is 1 +/- 0.1 with the min 0.889 and max 1.117.
- The mean of the features is 0 +/- 0.15 with the min -0.2 and max 0.1896.

More detailed results can be found in the attached jupyter notebook.

3 Difficulties and Challenges

From the analysis we did, the main characteristic of this dataset is that **the test set is much larger than the training set** and **each data point is with high dimensions**. This may lead to the following difficulties and challenges:

- Overfitting problem: As we know, the goal of a machine learning model is to generalize patterns in training data so that we can correctly predict new data that has never been presented to the model. Overfitting occurs when a model adjusts excessively to the training data, seeing patterns that do not exist, and consequently performing poorly in predicting new data.

- Feature engineering problem: For each data point, it has 300 features without practical meaning. Unlike some classification or regression problems based on real life situations, this dataset does not clearly indicate the meaning of the data points, which may result in difficulties in feature engineering.

4 Dealing with Overfitting: Cross-validation

From the lectures, we learned the cross-validation method and its advantages. The basic idea is to use the initial training data to generate multiple mini train-test splits and then use these splits to tune the models. Detailed explanation of the cross-validation are omitted here. We will also adopt this method to help preventing overfitting. We will use this method in the following section to do a preliminary experiments to take a look of different models.

5 Dealing with Overfitting: Simple Model

Large and complex models may have more parameters and lead to a extremely curved border which can almost explain every data point. We will then try to avoid complex models with many parameters, thus limiting their generalization and possibility of overfitting.

We will use the following models to do the preliminary experiments. For these models, we used the *sklearn* package. Here I will briefly introduce some of the models.

- **Decision tree:** Decision trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

- **Support vector machine (SVM):** The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points. It can be used in both classification and regression. The advantages of support vector machines is that it is effective in high dimensional spaces.

- **K-nearest neighbors (KNN):** In pattern recognition, the k-nearest neighbors algorithm is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

- **Random Forest:** Random forest is a commonly-used machine learning algorithm, which combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

- **AdaBoost:** AdaBoost is one of the first boosting algorithms to be adapted in solving practices. Adaboost helps to combine multiple “weak classifiers” into a single “strong classifier”. The weak learners in AdaBoost are decision trees with a single split, called decision stumps. AdaBoost works by putting more weight on difficult to classify instances and less on those already handled well.
- **Gradient Boosting:** Gradient Boosting is similar to AdaBoost in that they both use an ensemble of decision trees to predict a target label. However, unlike AdaBoost, the Gradient Boost trees have a depth larger than 1.
- **ExtraTrees:** The Extra Trees algorithm works by creating a large number of unpruned decision trees from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression or using majority voting in the case of classification.

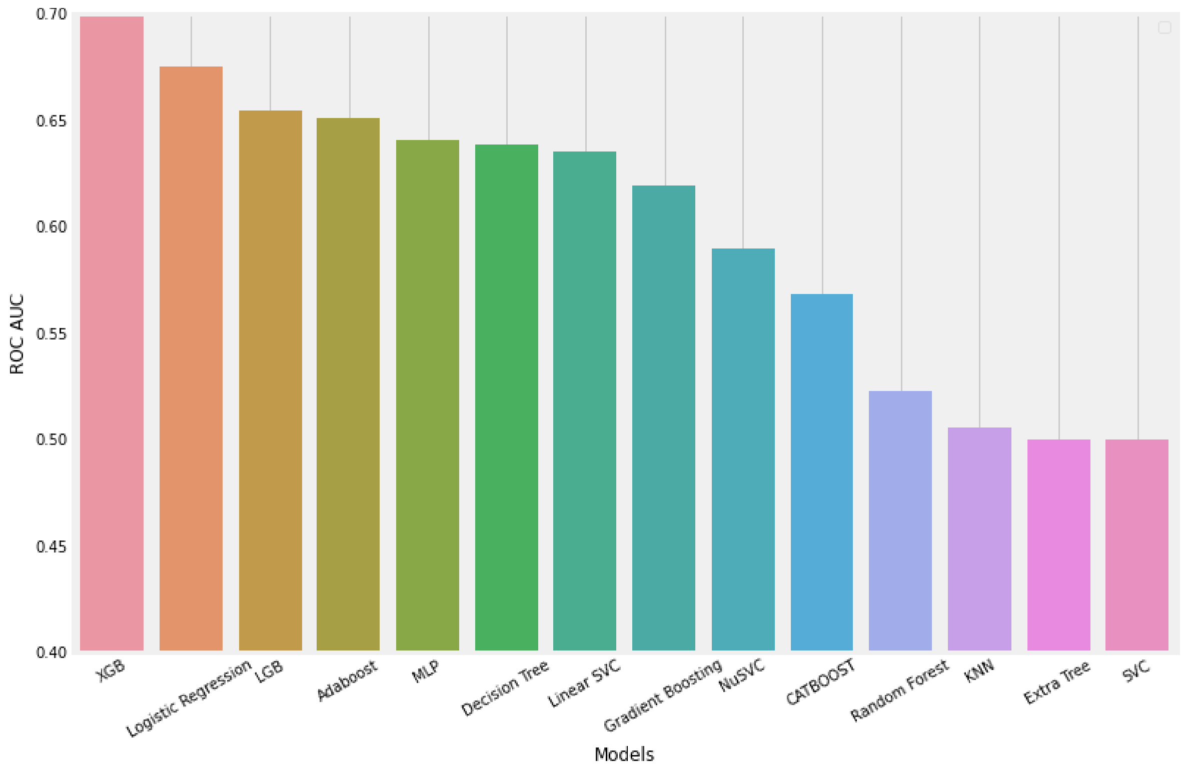


Figure 3: Preliminary experiments

By using the AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve as our metric, we get the preliminary experiment results shown in Figure 3. From the Figure 3, we can see that using XGB and Logistic Regression (LR) can achieve better performance. Unsurprisingly, MLP does not achieve very good performance. One possible reason is that the neural network models are more likely to overfit. On the contrary, simple models like LR can still have leading performance. We will then do the further hyperparameters adjusting and model combination using these two models in the following sections.

6 Dealing with Overfitting: Regularization

Regularization is a technique used for tuning the function by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values.

Therefore, in our model, we will also employ L1 and L2 regularization to help avoiding the overfitting problem. The formulas of L1 and L2 regularization are shown below:

$$Loss = \text{Error}(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i| \tag{1}$$

$$Loss = \text{Error}(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2 \tag{2}$$

7 Dealing with Overfitting: Outliers Elimination

Outliers elimination is also very crucial to prevent overfitting. On the one hand, with very limited training points (250 in total), even if there are only 50 outliers, they can already account for 20% of the training set. On the other hand, pruning the outliers can help to improve the models' generalization ability.

We will use the **IsolationForest** package in sklearn to detect the outliers. The **IsolationForest** 'isolates' observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node.

8 Dealing with Overfitting: Feature Selection

Feature selection improves the machine learning process and increases the predictive power of machine learning algorithms by selecting the most important variables and eliminating redundant and irrelevant features. Meanwhile, it can also make the training process shorter.

Here we will use **ExtraTreesClassifier** to help rank the features' importance. The purpose of the ExtraTreesClassifier is to fit a number of randomized decision trees to the data, and in this regard is a from of ensemble learning. Particularly, random splits of all observations are carried out to ensure that the model does not overfit the data.

9 Dealing with Overfitting: Label De-bias

When we are doing a classification task, if the dateset is imbalanced i.e. there are too few samples of the minority class, it may be difficult for the model to effectively learn the decision boundary

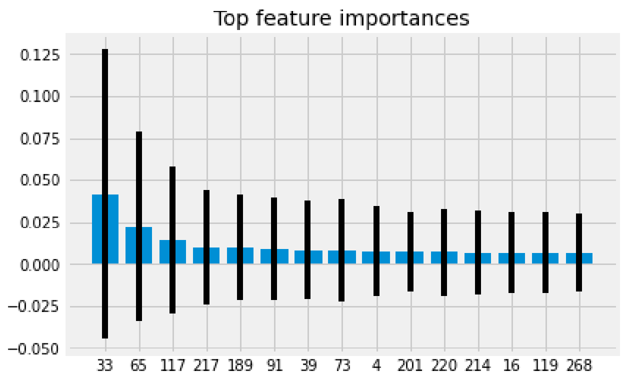
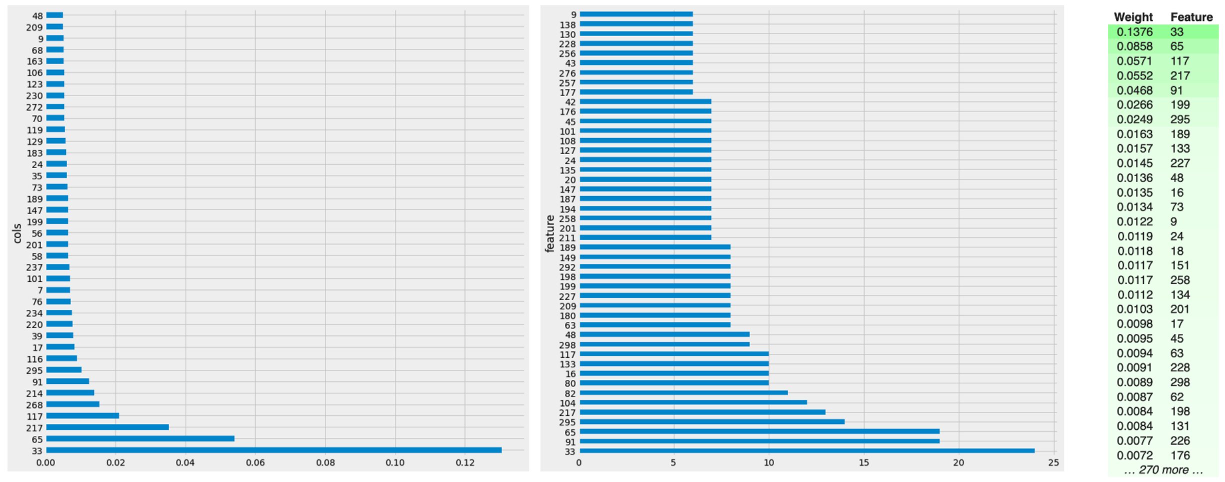


Figure 4: Top feature importance



Weight	Feature
0.1376	33
0.0858	65
0.0571	117
0.0552	217
0.0468	91
0.0266	199
0.0249	295
0.0163	189
0.0157	133
0.0145	227
0.0136	48
0.0135	16
0.0134	73
0.0122	9
0.0119	24
0.0118	18
0.0117	151
0.0117	258
0.0112	134
0.0103	201
0.0098	17
0.0095	45
0.0094	63
0.0091	228
0.0089	298
0.0087	62
0.0084	198
0.0084	131
0.0077	226
0.0072	176
… 270 more …	

Figure 5: Feature importance ranking results using different methods

One way to solve this problem is to oversample the examples in the minority class. This can be achieved by simply duplicating examples from the minority class in the training dataset prior to fitting a model. This can balance the class distribution but does not provide any additional information to the model.

An improvement on duplicating examples from the minority class is to synthesize new examples from the minority class. This is a type of data augmentation for tabular data and can be very effective.

One of most popular algorithm to do the oversampling might be the Synthetic Minority Oversampling TEchnique [1], or SMOTE for short. SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.

Specifically, a random example from the minority class is first chosen. Then k of the nearest neighbors for that example are found (typically k=5). A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space.

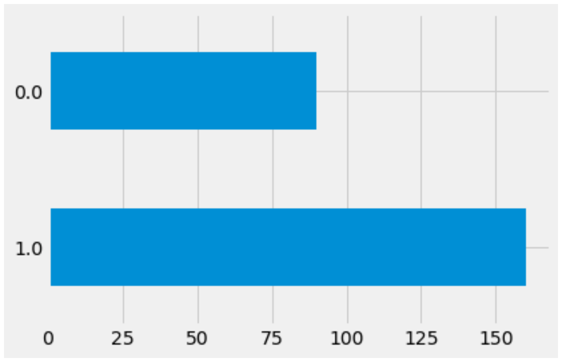


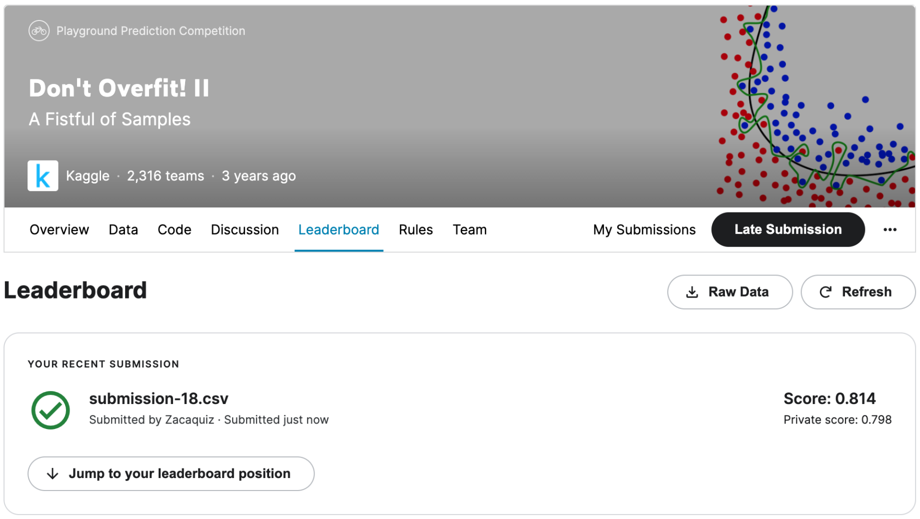
Figure 6: Label distribution

10 Dealing with Overfitting: Models Combination

Similar to the idea of Bagging (also known as bootstrap aggregation), combining the weak models may lead to better performance by reducing the variance and enhancing the generalization. From the preliminary experiments, we will choose XGB and logistic regression as our baseline models and try to find the weighting of each estimator.

The tool we will use here is **sklearn.ensemble.StackingClassifier**. Stacked generalization consists in stacking the output of individual estimator and use a classifier to compute the final prediction. Stacking allows to use the strength of each individual estimator by using their output as input of a final estimator.

11 Final Result



Don't Overfit! II

A Fistful of Samples

Kaggle · 2,316 teams · 3 years ago

Leaderboard

YOUR RECENT SUBMISSION

submission-18.csv

Submitted by Zacaquiz · Submitted just now

Score: 0.814

Private score: 0.798

Figure 7: Final result

12 Ablation Test

After applying all of the above techniques to avoid overfitting, one natural question is: do they really contribute in preventing overfitting? If so, which one contribute the most? To answer these questions, we did the ablation test to prove the efficiency of each technique. Noticed that if we drop cross-validation method, we will get different results in Figure 3 thus influence the final result.

Excluded Features	Accuracy
Regularization	0.756
Outliers Elimination	0.778
Feature Selection	0.749
Label De-bias	0.779
Models Combination	0.750
Cross-validation	0.698

Table 1: Ablation test result

From the ablation test, we can find all the methods we employ here contribute to the generalization of the model.

13 Conclusion

In this report, we focus on the methods and results of the Kaggle competition: Don't overfit II. We applied seven different methods to prevent overfitting and achieved great performance in the test set. The ablation test shows the effectiveness of the techniques we employed here.

Appendices

References

[1] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.