COMP4432 Machine Learning

Kaggle Competition Report: Don’t Overfit II

Siyu Zhou (21094655D) | November 13th, 2024

1. Introduction

The report basically includes the data science competition from Kaggle, Don’t Overfit II Competition [1], emphasizing the importance of overfitting problem. The objective of this report is to introduce various methods to address overfitting problems, by analyzing dataset, choosing different models, utilizing different training methods, evaluating model performance, and analyzing improved methods & improved models. The analysis and method mentioned in this report is implemented in the notebook link in *Appendix 1*.

2. Dataset Overview

2.1 Statistical Analysis

The Don’t Overfit dataset for this report is from the **older version** [2] as new data version will encounter evaluation issues. The training dataset consists of 250 samples. Each sample has 300 continuous features with a binary target variable $target$ a sample index id from 0 to 249. Also, the testing dataset has 19750 samples, and each sample has the same structure as the training dataset except the $target$ variables.

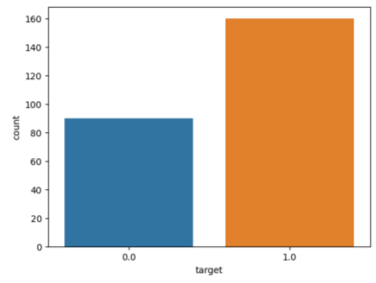


Figure 1 Distribution of Target value in training dataset

In the training dataset, there’s no missing data, and no duplicate data. And it consists of 160 samples with $target$ value as **1**, and 90 samples’ $target$ values are **0**.

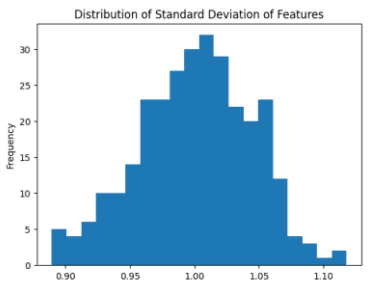


Figure 2 Distribution of Standard Deviation of Features

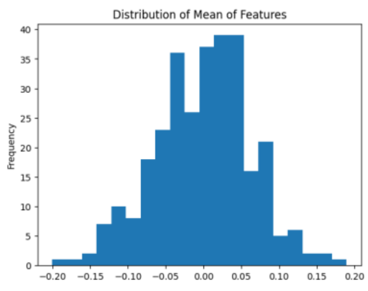


Figure 3 Distribution of Mean of Features

By observing $Figure$ 1, $Figure$ 2, $Figure$ 3, it is found that:

1. In the training dataset, the number of attributes is greater than number of samples, which may lead to overfitting problems.
2. In the training dataset, the number of samples with $target$ value 1 is greater than sample number with $target$ 0, which may lead to high bias in unseen testing dataset.
3. Features in the training dataset have a standard deviation of 1 +/- 0.1 (minimum and maximum values are 0.889, 1.117 respectively).
4. Features in the training dataset have a mean of 0 +/- 0.15 (minimum and maximum values are -0.2, 0.1896 respectively).

2.2 Feature Distribution Analysis

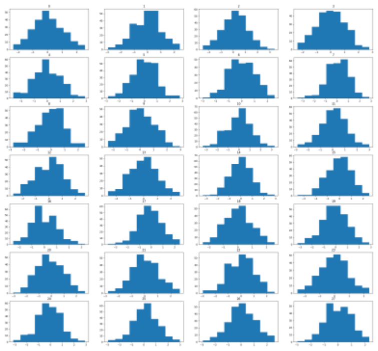


Figure 4 Distribution of First 28 columns in training dataset

By observing distribution of random selected 28 columns from training data shown in $Figure$ 4, it is found that:

- Distribution of values in columns are similar.
- For each attribute, the value among samples is almost in normal distribution.

2.3 Feature Correlation Analysis

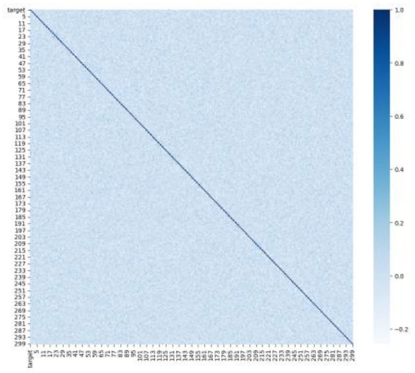


Figure 5 Correlation between Features and Target

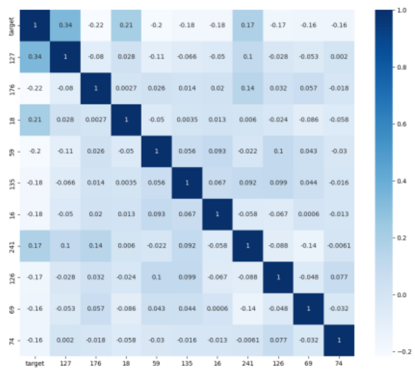


Figure 6 Correlation between Top 10 features and Target

By observing correlations between features & target in $Figure$ 5 and $Figure$ 6, it is found that:

- There's no strong correlation among features, and no strong relation between any features.
- There is no obvious correlation between features and target, and the importance of any features is not outstanding.

In summary, by data analysis, the **high entropy** (imbalance) is found in number of attributes and sample numbers in training dataset, the number of samples between training and testing dataset, and the number of samples with $target$ 0 and 1. Comparatively, the **low entropy** is in distribution among samples and features, correlations among features, and correlations between features and target.

3. Basic Feature Engineering Methods

3.1 Data Cleaning & Preparation

The training dataset is relatively clean and contains no missing data and no duplicated data. Then the column id is removed from the training dataset.

3.2 Normalization: Standard Scalar

Feature normalization is performed through scaling using $StandardScalar$ function [3] from scikit-learn python library. Normalization is important for the train model, since it can enable feature value to be scaled, and fall within a small and controllable range, which is crucial for models like SVM and Logistic Regression. The transformation is shown as below:

$$z = StandardScaler(x) = \frac{x-\mu}{\sigma} [3]$$

In this formular, x represents the original features, μ represents the mean value of the features, and σ is the standard deviation of the feature.

4. Basic Modelling Methods

4.1 Evaluation Metrics - AUCROC

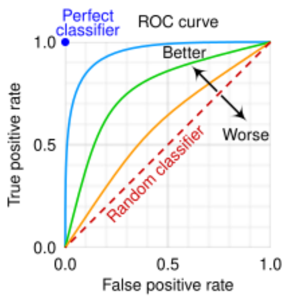


Figure 7 ROC space for Better Or Worse classifier [4]

According to the Kaggle Competition Overview, the submission result is evaluated by AUCROC metrics between predicted value and the true labels for performance. The closer the AUC-ROC score is to 1, the better the model is at predicting the positive class. An AUC score of 0.5 suggests no discriminative ability, akin to random guessing. Since the thresholds are only 0 and 1, the AUCROC result may being low.

In following implementation, we ensure that models used have either $predict\_proba$ or $decision\_function$ for more meaningful AUC-ROC evaluation. Without probabilities or scores, $predict()$ will give class labels, which are not ideal for AUC metrics.

4.2 Basic Models

In this part, different kinds of basic models are introduced for the competition together. The performances of these models are evaluated with AUCROC score. It is more effective to find more potential models with higher performance on testing datasets through trying various models. We chose 9 basic models here, including as follows [5]:

- $\boldsymbol{Logistic\ Regression}$: a linear model that is easy to interpret and works well for linearly separable datasets. Adding class weights helps to deal with imbalanced data.
- $\boldsymbol{SVM\ Classifier}$: Support Vector Classifier with an RBF kernel is suitable for complex decision boundaries. The `C` parameter controls the trade-off between a smooth decision boundary and classifying training points correctly.
- $\boldsymbol{KNN\ Classifier}$**:** a simple and effective algorithm for smaller datasets. The choice of $n\_neighbors$ can impact model performance significantly.

- ***Decision Tree***: a simple and interpretable model that splits data based on feature importance. However, it can easily overfit if not pruned.
- ***Gaussian Naive Bayes***: a probabilistic classifier that assumes normal distribution of features. It is fast and works well for text classification and problems with categorical features.
- ***AdaBoost***: an ensemble method that builds multiple weak learners sequentially. It works well to improve accuracy for weak base models like decision trees.
- ***Stochastic Gradient Descent***: a simple yet efficient method for large-scale linear classification. It can be sensitive to feature scaling, which is why standardization is important .
- ***Lasso*** (L1-regularization): a linear model that helps in feature selection by shrinking less important features' coefficients to zero, which is suitable for high-dimensional data.
- ***Random Forest***: an ensemble learning method that uses multiple decision trees to improve model robustness and reduce overfitting. This method is effective in identifying key features and reducing dimensionality.

By splitting scaled training datasets into the training and validation part, we could investigate it as a preview of the performance of these 9 different models as the following figures show:

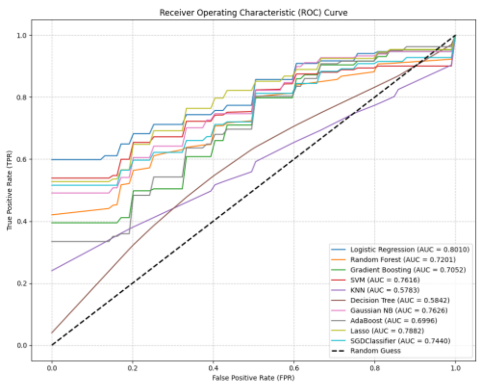


Figure 8 ROC Curve for different models

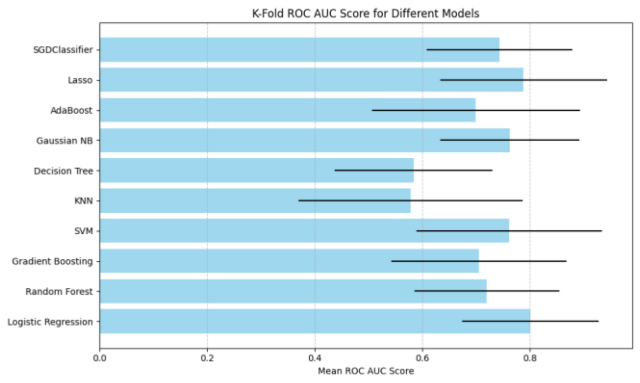


Figure 9 ROC AUC Score for different Models

By Observing the above *Figure* 8 and *Figure* 9, it shows that:

- *Logistic Regression* (LR) Model and *Lasso* Model have relatively higher performance (ROC AUC Score) than other models.

- *Support Vector Machine* (SVM) Classifier and *K Nearest Neighbor* (KNN) Classifier have relatively lower performance (ROC AUC score) than other models.

4.3 Cross-Validation

For K Fold Cross Validation Part, we used *StratifiedKFold* function for training models. Through 20-fold cross-validation strategy, the dataset is randomly divided into 20 equal parts (folds). In each training iteration, 19 folds are used for training mode, then the left fold is used for validation. And each fold is used once for validation in each process.

For fold number, parameter number, and regularization type, the validation is shown by AUCROC score as following table. The dataset is from scaled training dataset and using logistic model.

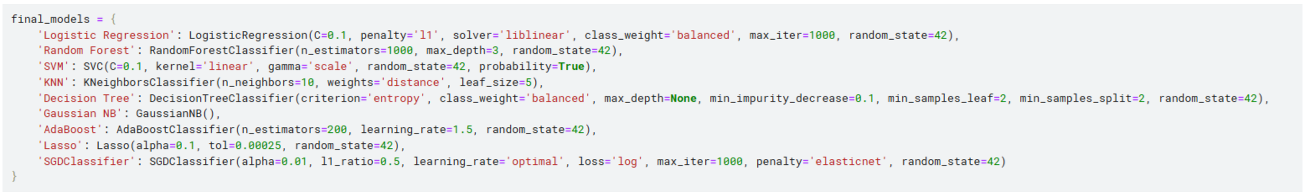
2-Fold CV			**5-Fold CV**			**10-Fold CV**			**20-Fold CV**		
	L1	L2		L1	L2		L1	L2		L1	L2
C=0.01	0.5000	0.7147	C=0.01	0.5000	0.7976	C=0.01	0.5000	0.7694	C=0.01	0.5000	0.7841
C=0.1	0.7181	0.7056	C=0.1	0.7736	0.7854	C=0.1	**0.8049**	0.7708	C=0.1	**0.8238**	0.7950
C=1	**0.7244**	0.7011	C=1	**0.7997**	0.7681	C=1	0.7819	0.7681	C=1	0.8094	0.7863

Table 1 ROC AUC score for Logistic Models with Different Parameter for Training Dataset

By observing the Table 1, we can find that the performance differentiates by changing parameters. More fold implemented from 2-fold to 20-fold, the score is higher. And when C=0.1, the AUCROC score would be better than model with c = 0.01. Also, $L1$'s performance is always better than that of $L2$. These parameters would be tuned by using GridSearchCV function in the next section.

4.4 GridSearchCV for Hyperparameter Tuning

Each model needs to be parameterized before training, which has a significant impact on the model's performance. To determine the optimal parameters for a model, we use the $GridSearchCV$ function, a powerful automation tool that tunes the parameters by cross-validating the model with different combinations of parameters [6]. Examples include the optimal number of clusters for the kNN model, the optimal partitioning of samples and features for the random forest model, and the optimal regularization parameters for the regularization model. The $AUCROC\ score$ was also used as an evaluation metric. The following model parameters are applied as $Figure$ 10 shows.



```
final_models = {
    'Logistic Regression': LogisticRegression(C=0.1, penalty='l1', solver='liblinear', class_weight='balanced', max_iter=1000, random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=1000, max_depth=3, random_state=42),
    'SVM': SVC(C=0.1, kernel='linear', gamma='scale', random_state=42, probability=True),
    'KNN': KNeighborsClassifier(n_neighbors=10, weights='distance', leaf_size=5),
    'Decision Tree': DecisionTreeClassifier(criterion='entropy', class_weight='balanced', max_depth=None, min_impurity_decrease=0.1, min_samples_leaf=2, min_samples_split=2, random_state=42),
    'Gaussian NB': GaussianNB(),
    'AdaBoost': AdaBoostClassifier(n_estimators=200, learning_rate=1.5, random_state=42),
    'Lasso': Lasso(alpha=0.1, tol=0.00025, random_state=42),
    'SGDClassifier': SGDClassifier(alpha=0.01, l1_ratio=0.5, learning_rate='optimal', loss='log', max_iter=1000, penalty='elasticnet', random_state=42)
}
```

Figure 10 Model and Model Parameter Applied

4.5 Basic Model Performance

After applying the strategies mentioned above, the predicted target results of 9 models are implemented. The leaderboard scores of basic models are shown in $Figure$ 11. $Logistic\ Regression$ and SVM performed reasonably well with scores of $\mathbf{0.753}$ and $\mathbf{0.629}$, respectively.

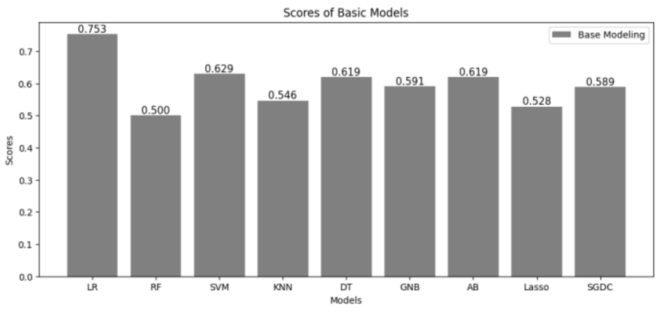


Figure 11 Leaderboard Score of Basic Models

5. Improved Methods

Based on observations in data analysis, the distribution of features of training dataset is normally distributed. It is not easy to spot any relation between features and target. Then, the improvement of model performance is not only based on the model selection and training with different parameters. Further feature engineering methods should be implemented, such as feature creation and feature selection.

5.1 Balance Samples – SMOTE method

As the imbalance of samples in two classes observed in Figure 1, we used $SMOTE$ technique to solve this problem. Synthetic Minority Oversampling Techniques (SMOTE) is used to generate samples for minority classes, then prevent overfitting problems from majority classes [7].

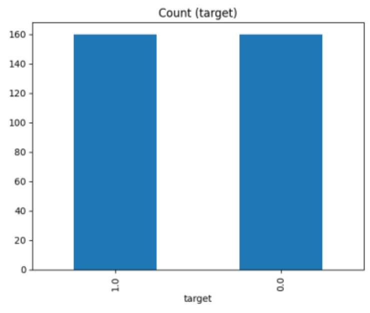


Figure 12 Distribution after SMOTE

5.2 Create Statistic Features: mean, standard deviation

For each sample, calculate the mean and standard deviation of the feature values, and add them as new features, resulting in 302 features.

5.3 Feature Selection

Since the number of attributes is larger than the number of samples in the training dataset, the dimensionality of the feature space can be reduced by selecting the most important features, which may help to prevent overfitting of the model to the training dataset. Various feature engineering techniques were employed to enhance model performance. The feature selection methods used include $SelectKBest$, $Recursive\ Feature\ Elimination$ (RFE), and $ELI5$-based Permutation Importance. These techniques were applied to reduce dimensionality and retain only important features, which, in turn, helped improve model performance, robustness, and reduce overfitting.

5.3.1 SelectKBest

$\boldsymbol{SelectKBest}$ selects the top 10 features based on the ANOVA F-value. This method helps to retain features that are most strongly correlated with the target variable, which improves the model's ability to capture significant patterns in the data [8]. Compared to the basic models, using $SelectKBest$ function improved the leaderboard scores for most models, especially SVM, KNN, and SGDC. The SVM model's leaderboard score increased from $\mathbf{0.629}$ $(\boldsymbol{basic})$ to $\mathbf{0.688}$ $(\boldsymbol{KBEST})$, showing a notable improvement in performance.

5.3.2 Recursive Feature Elimination

$\boldsymbol{Recursive\ Feature\ Elimination}$ (RFE): applied using a $LogisticRegression$ estimator to recursively remove less important features until only the **top 10 features** remained. This method ensures that only the most predictive features are used for modeling, which can lead to simpler and more interpretable models [9]. The performance with RFE was generally lower compared to other feature selection methods, with leaderboard scores for models like LR, RF, and SVM being lower than basic ones. This indicates that RFE might have removed a few important features, leading to a reduction in model effectiveness.

5.3.3 ELI5-based Permutation Importance

Permutation importance was used to evaluate feature importance by shuffling individual features and assessing the drop in model performance. The top 10 features were selected based on their contribution to the model's performance [5][10]. This method showed improvements in model scores, especially for LR, SVM, and SGDC. The *LogisticRegression* model, for example, had a score of $\mathbf{0.717}$ ***with ELI5***, compared to 0.753 *for baseline*. Although the improvement was not as significant as with SelectKBest, it still demonstrated that the selected features contributed positively to the model's predictive power.

5.4 Improved Model Performance

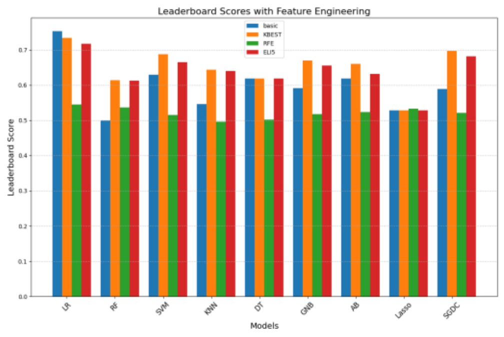


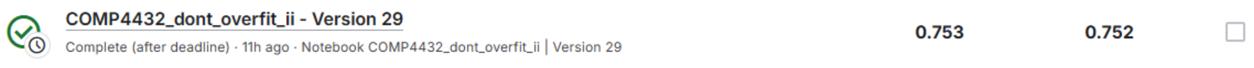
Figure 13 Leaderboard Score with different feature engineering mtheods

Most Feature Creation methods have less or negative effects on the performance, and most of Feature Selection methods have a positive effect, where *ELI5 − based Permutation Importance* method contributes the most, as shown in the *Figure* 12.

In conclusion, feature selection methods play a crucial role in improving model performance by ensuring that only the most relevant features are used. *SelectKBest* and *ELI5* provided the **most significant** boosts in performance, while *RFE* was less effective. ***Logistic Regression***, ***SVM***, and ***SGDClassifier*** consistently performed well with feature engineering, while models like ***Random Forest*** and ***Decision Tree*** showed less improvement, suggesting that these models may benefit more from hyperparameter tuning than feature selection.

6. Conclusion

Team Name: Siyu Zhou



COMP4432_dont_overfit_ii - Version 29
Complete (after deadline) · 11h ago · Notebook COMP4432_dont_overfit_ii | Version 29
0.753 0.752

In this competition, we performed data analysis, feature engineering, model training with some strategies, and several improvement methods. The best leaderboard score is 0.753 with LR base model. Although improvement is not effective on LR, it is still good practice to address overfitting problems with other models such as Logistic Regression, SVM, and SGDClassifier.

Reference

[1] P. Culliton, “Don’t Overfit! II,” *Kaggle.com*, 2019. https://www.kaggle.com/competitions/dont-overfit-ii/overview (accessed Nov. 12, 2024).

[2] M. Mubasir, “Older Dataset for Don’t overfit II challenge,” *Kaggle.com*, 2020. https://www.kaggle.com/datasets/mdmub0587/older-dataset-for-dont-overfit-ii-challenge (accessed Nov. 12, 2024).

[3] Scikit-Learn, “sklearn.preprocessing.StandardScaler — scikit-learn 0.21.2 documentation,” *Scikit-learn.org*, 2019. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[4] Wikipedia Contributors, “Receiver operating characteristic,” *Wikipedia*, Mar. 20, 2019. https://en.wikipedia.org/wiki/Receiver_operating_characteristic (accessed Nov. 12, 2024).

[5] artgor, “How to not overfit?,” *Kaggle.com*, Apr. 24, 2019. https://www.kaggle.com/code/artgor/how-to-not-overfit/ (accessed Nov. 12, 2024).

[6] Scikit-Learn, “GridSearchCV,” *scikit-learn*, 2024. https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.GridSearchCV.html

[7] R. Alencar, “Resampling strategies for imbalanced datasets,” *kaggle.com*. https://www.kaggle.com/code/rafjaa/resampling-strategies-for-imbalanced-datasets (accessed Nov. 13, 2024).

[8] Scikit-Learn, “SelectKBest,” *scikit-learn*, 2024. https://scikit-learn.org/dev/modules/generated/sklearn.feature_selection.SelectKBest.html (accessed Nov. 13, 2024).

[9] Scikit-Learn, “RFE,” *scikit-learn*, 2024. https://scikit-learn.org/dev/modules/generated/sklearn.feature_selection.RFE.html

[10]M. Korobov and K. Lopuhin, “Permutation Importance — ELI5 0.11.0 documentation,” *eli5.readthedocs.io*. https://eli5.readthedocs.io/en/latest/blackbox/permutation_importance.html (accessed Nov. 13, 2024).

Appendix

1. Notebook Link: https://www.kaggle.com/code/siyuuzoeyzhou/comp4432-dont-overfit-ii