# Model Ensemble is Powerful

*Yihan Zhang 1823149*

***Introduction –*** Here I will introduce the rationale of the random forest classifier model which gains the highest accuracy **(0.90476)** in Kaggle public leaderboard and discuss its corresponding parameters in sklearn package. Random forest is a type of ensemble learning, forest means that it has many decision trees, random means that each decision tree is trained based on different samples and features so that it can reduce the influence of outliers. Finally, the result will be generated by the voting among these decision trees.

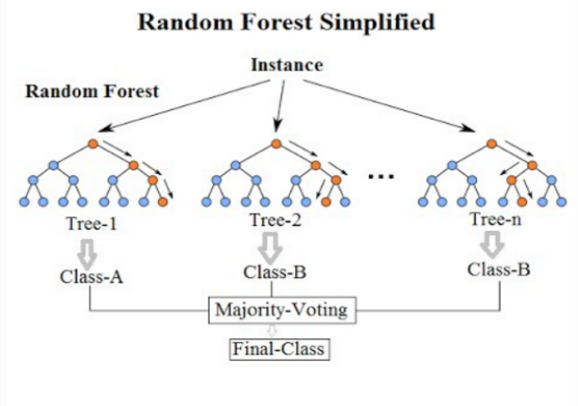


Figure 1: Structure of Random Forest Model

'n_estimater' is the number of the trees in the forest, default value is 100, and the underfitting can be happened if this parameter is small. 'max_depth' is the depth of each tree, default is null, which means there is no limit of the depth.

***WorkFlow of Framework –*** Firstly, we need to clean the original data, including drop duplicated data, fill null values, eliminate some useless attributes to reduce the dimension, and the feature selection can be implemented by random forest. As for categorical data, we need to do One Hot encoding though in this dataset, it has already done this part. For numerical data which has large skewness, we need to do the data normalization so that the model can be converged quickly. Then we need to choose suitable models, here I have tried several models including KNN (k-nearest neighbors), Random Forest, AdaBoost (Adaptive Boosting) and XGBoost (eXtreme Gradient Boosting) models. Fine tune the parameters will be done in the final step. Finally, the testing results file using different models will be generated.

***Results –*** I have tried random forest, AdaBoost and XGBoost models and their parameters are tuned by GridSearchCV package, which will do the permutation and combination on different parameters and choose the best combination of parameters. It is obvious to find that data normalization can enhance the performance and it is interesting to find that the complicated model, XGBoost and AdaBoost is not better than the Random Forest model. The reason might be their parameters are hard to tune.

| Model | Parameter | Accuracy |
|---|---|---|
| *Random Forest with data normalization* | n_estimators=100, max_depth=4, criterion="gini" | 0.90476 |
| *Random Forest without data normalization* | n_estimators=100, max_depth=4, criterion="gini" | 0.80952 |
| *AdaBoost with normalization* | N_estimators=100, learning_rate=0.1 | 0.85714 |
| *AdaBoost without normalization* | N_estimators=100, learning_rate=0.1 | 0.76190 |
| *XGBoost with normalization* | learning_rate=0.2, n_estimators=50, max_depth=5, objective='binary:logistic', colsample_bytree=0.8, subsample=0.8, use_label_encoder=False | 0.80952 |
| *XGBoost with normalization* | learning_rate=0.2, n_estimators=50, max_depth=5, objective='binary:logistic', colsample_bytree=0.8, subsample=0.8, use_label_encoder=False | 0.80952 |

Table 1: Comparison of Accuracy on Kaggle among different models with or not data normalization

***Pros and Cons of Random Forest –***

**Pros1:** Can output which attributes are important.
**Pros2:** Reduce the influence of outliers.
**Cons1:** Spend more time to train many trees compared with decision tree model.

I think another advantage of Random Forest model is that it is easy to understand compared with complicated models. In order to improve the model performance, we can utilize some packages such as GridSearchCV to choose the optimal parameters.

***Conclusion and Personal Experience –*** When facing a classification problem, we need to consider all of aspects that can influence the result. A good model is necessary, but a complicated model is not always the best model that can make you successful and it could be more time consuming to tune its parameters, which is not efficient. Even simple models can achieve better performance with simple tuning. Through this Kaggle competition, I find that concentrating more on how to do the data preprocessing is more significant compared with choosing a model. Finally, tuning the hyperparameters is important because sometimes the model can be overfitting so that we assume we have high rank in the public leaderboard but get a low rank in the private leaderboard.