CompSci 671 Machine Learning

Kaggle Machine Learning Competition Report

Minghui Zhu (mz223)

December 9th, 2022

In this article, the first section describes an exploratory analysis in which I preprocess data and make sense of data and features. I also display the feature selection to drop unimportant features and improve accuracy. In the following sections, I present the evaluated models, their selection criteria, the data-splitting, and the hyperparameter tuning. Finally, I report the prediction result with a summary. The code is attached as the appendix of the result and submitted to Sakai.

1 Exploratory Analysis

1.1 Dataset and Task Overview

This dataset contains employee and company data. The training dataset has 1,340 observations with 35 features, including age, gender, education, distance from home, etc. The task is to apply multiple machine learning algorithms to produce classification predictions on the attribute "Attrition," a binary variable indicating whether an employee left the company. The evaluation criterion is the F1-score.

1.2 Data Preprocessing

To begin with, I verify the missing data and duplicated data issues, and luckily neither the training dataset nor the testing dataset include any missing data or duplicated data.

The training dataset has 9 categorical and 26 continuous variables. Since machine learning models require numeric input and output variables, I must encode the categorical features. I assigned numeric labels (0, 1, 2…) to each ordinal variable and make dummies for nominal variables. At this stage, I eliminate 3 variables, 'EmployeeCount,' 'Over18', and 'StandardHours,' which have the same unique value for all observations and therefore make no sense in analysis.

1.3 Correlation Check

I compute the pair correlation between each pair of variables and plot the results in Figure 1. A correlation matrix is a table depicting the correlation between all possible pairs of correlation coefficients. I specify the legend to emphasize both positive and negative correlation coefficients: the cell in the table tends to be transparently white if the correlation coefficient between two attributes is close to 0. When two features have a significant positive correlation, the cell tends to be a vibrant shade of green, while the color changes to pink when strongly negative.

Specifically, we examine all the cells in the first column (or the first row, they are symmetrically identical), which is the correlation between the target variable “attrition” and every other feature. Gender and the number of companies worked are uncorrelated to the attrition rate. Age, employment level, monthly salary, and marital status are inversely connected with the attrition rate, whereas overtime increases the likelihood of employees leaving the company.

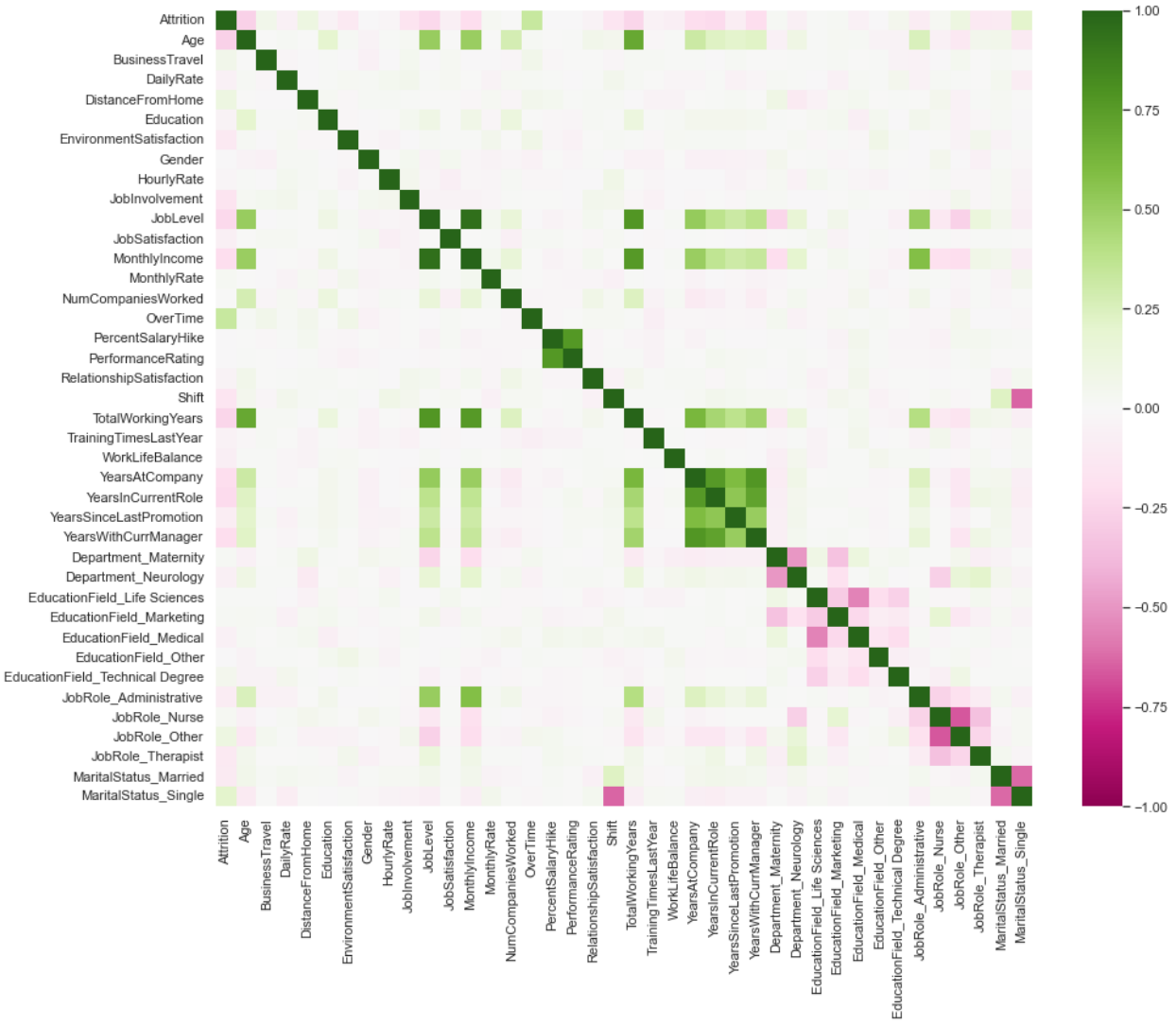


Figure 1: Correlation Matrix

In addition, severe multicollinearity exists between multiple pairs of variables when the cell is dark green, which means a strong correlation exists. For instance, there is a correlation between employment level and monthly income. 'YearsAtCompany' is highly correlated with 'YearInCurrentRole', 'YearsSinceLastPromotion', and 'YearsWithCurrManager'. I discard certain features to address the multicollinearity issue.

1.4 Imbalanced Data and Feature Engineering

To know more details about the dataset, Figure 2 draws histograms that present the distribution of each feature. From Figure 1, we can see that the dataset is highly imbalanced. A bunch of features is moderately skewed where the proportion of the minority group is less than 20%. Hence, the imbalance issue must be considered when choosing machine learning models. Besides, it is easy to find that some features are distributed as long-tailed, inspiring me to conduct feature engineering to take log transformation and magnify the effect. Further modeling results tell it does help.

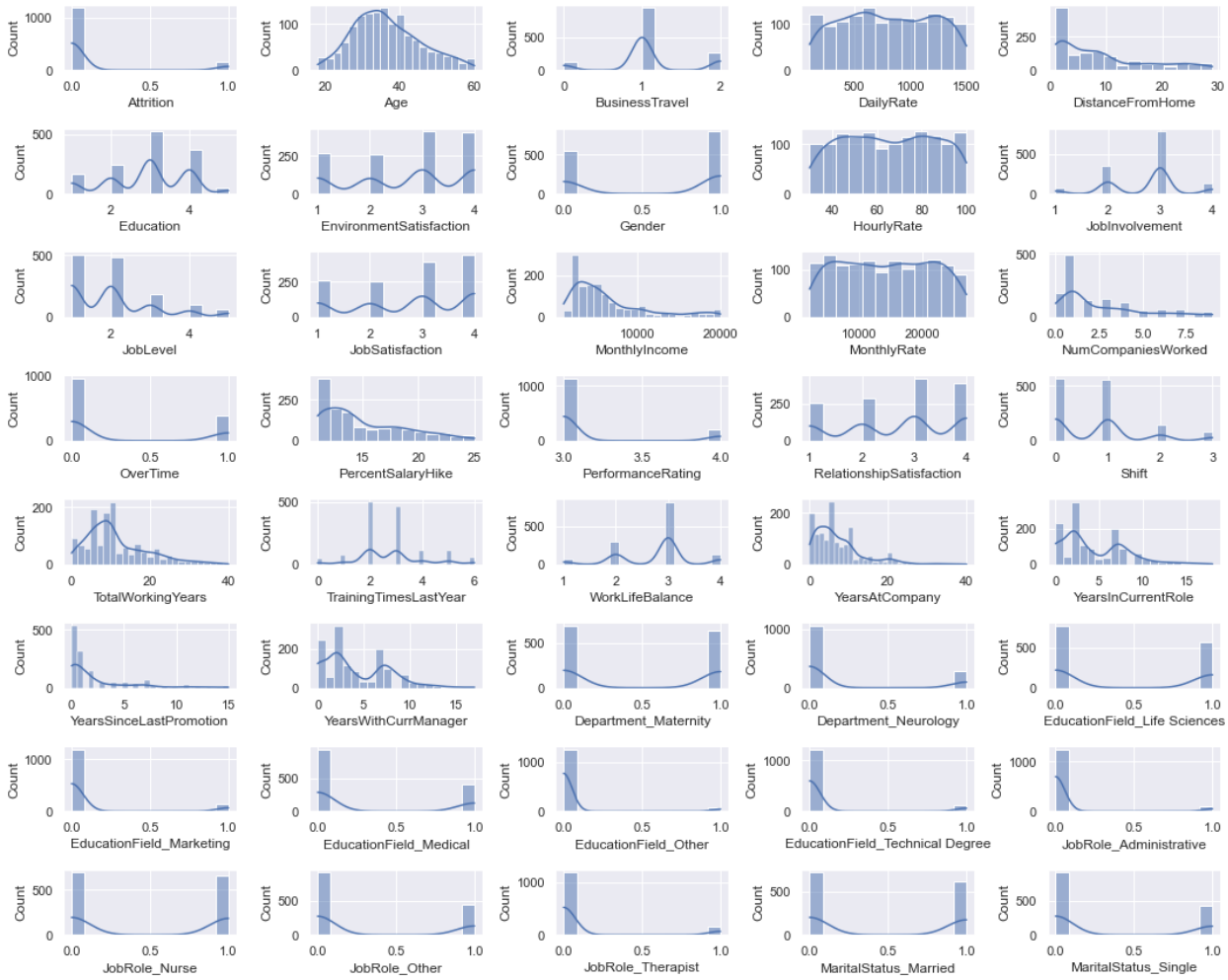


Figure 2: Distribution of Each Feature

1.5 Feature Selection

There are still 31 features in the training dataset, but I will not continue to make feature selections by feature importance.

I used to apply Recursive Feature Elimination with Cross-Validation (RFECV)[1] to identify the optimal number of features and the essential features. RFE is a feature selection method that fits a model, ranks the features, and recursively deletes the least important features and re-fitting the model. I choose Random Forest Classifier as the base estimator. However, the feature subset selected performs worse than putting all the remaining 31 features as a whole, especially when I fit a logistic regression. It may be due to the imbalanced data issue.

What's more, the other models I choose are tree-based, so there is no necessity to include additional feature selection as the tree models like Random Forest will automatically eliminate unimportant features and improve accuracy. Hence, I will apply the remaining 31 features to fit models.

2 Models

Considering the imbalanced nature of the dataset, I choose three models to fit: logistic regression with balanced class weight, balanced random forest classifier, and balanced bagging classifier.

Standard machine learning algorithms are not used because they assume that data is evenly distributed within classes. The model has enough data to learn for the majority group. Still, much fewer data to know about the minority group leads to a biased prediction and high misclassification

errors for the minority group. Hence, I apply algorithms with a specialty to deal with imbalanced data.

2.1 Logistic regression with balanced class weight

The first model I uses is logistic regression with balanced class weight. Logistic regression is commonly used when classifying a binary variable, like "Attrition," in my case. Logistic regression is easy and fast to train and implement. It also provides good interpretability. I use "sklearn.linear_model.LogisticRegression" from the "sklearn" library and specify the parameter "class_weight = 'balanced.'

2.2 Balanced Random Forest Classifier

The second model is the Balanced Random Forest Classifier. Random Forest is popular since it is more accurate than the decision tree models and solves the overfitting issues in decision trees. It is easy to implement in many libraries and can produce a valid prediction without tuning the hyperparameters. Based on the advantages of random forest, the balanced Random Forest classifier modifies it to be suitable for imbalanced data[2]. It creates two bootstrapped sets for each tree, one for the minority class and the other for the majority class, both of which are the same size and correspond to the size of the minority class. Balanced random forest gives a more effective performance. I use "imblearn.ensemble.BalancedRandomForestClassifier" from the "imblearn" library.

2.3 Balanced Bagging Classifier

The third model I use is the balanced bagging classifier. Bagging has the benefit of allowing numerous weak learners to work together to outperform a single good learner. It also aids in the decrease of variance, hence avoiding model overfitting in the method. However, the regular bagging classifiers generate numerous estimators on distinct random subsets of data, which do not allow us to balance each data group. As a result, when we train our model on an imbalanced dataset, the majority class will be favored[3]. Using balanced bagging classifier help alleviate the issue, which can work well in our task. I choose Hist Gradient Boosting Classifier as the base estimator, which has the advantages of boosting tree models like decreasing the variance and bias but performs faster than other boost models like AdaBoost. These models are available in Python library. I use "imblearn.ensemble.BalancedBaggingClassifier" from the "imblearn" library and the "sklearn.ensemble.HistGradientBoostingClassifier" fro the "sklearn" library.

2.4 Voting Classifier

A voting classifier is a machine learning estimator that trains many base models or estimators and predicts by aggregating their results[4]. It is available in "sklearn.ensemble.VotingClassifier". I will combine my previous models to see whether there is an improvement in the prediction.

Finally, I do not choose robust classifiers such as support vector machine (SVM) and deep neural networks (DNN). Since our data volume is tiny, these models are more likely to be overfitting on the training set.

3 Training

I set up a function to perform the training-fitting-evaluation process. A stratified k-fold cross-validation is used to adjust hyperparameters. I divide the train data set into 5 different folds and iterated through all combinations of training on nine folds and testing on the remaining fold. The F1 score is calculated and taken average and will be compared between the combinations.

Specifically, I choose to tune “solver” and “C” (inverse of regularization strength) in logistic regression to which algorithm and regularization strength will optimize the accuracy and avoid overfitting. For the balanced random forest model, the number of estimator and max_features are important because it is highly correlated with prediction performance. min_sample_leaf is also tuned since they it determines the ability to capture noise in the data. Finally, in the balanced bagging model, the learning rate is tuned because it sets the step size at each iteration while attempting to minimize a loss function. A small learning rate will slow the learning process, and a large one might make the process to too-fast and unstable.

When I train models and conduct hyperparameter selection, I connect it to a Google spreadsheet in the Google Cloud platform to record the details. It helps!

4 Hyperparameter Selection

I use “GridSearchCV” from the “sklearn” library to conduct hyperparameter tuning. GridSearchCV will automatically train the model with each parameter pair in the parameter grid, then it will evaluate the model with cross-validation to get a more robust measurement of the prediction accuracy. According to the evaluation standards in Kaggle and the imbalanced data nature, F1-score is specified at the scoring parameter. I tune only one parameter in each time, so I can record and plot the relationship between the F1 score and varying parameters. The plotting method is adopted from open resources[5].

4.1 Logistic Regression with Balanced Class Weight

For my logistic regression model, the hyperparameters I use are the “solver”, and “C” (inverse of regularization strength). I change the solver from ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'], where the ‘newton-cg’ performs best. I tune the C from 0.0001 to 1000 in 50 values. The best C is 0.39069. The graphs are below.

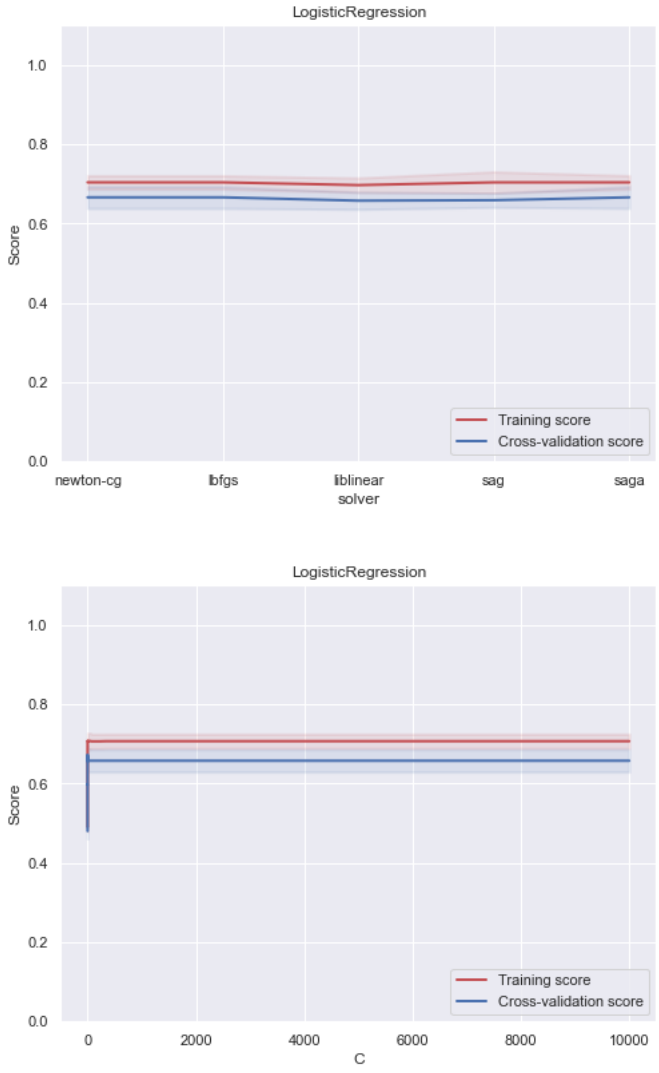
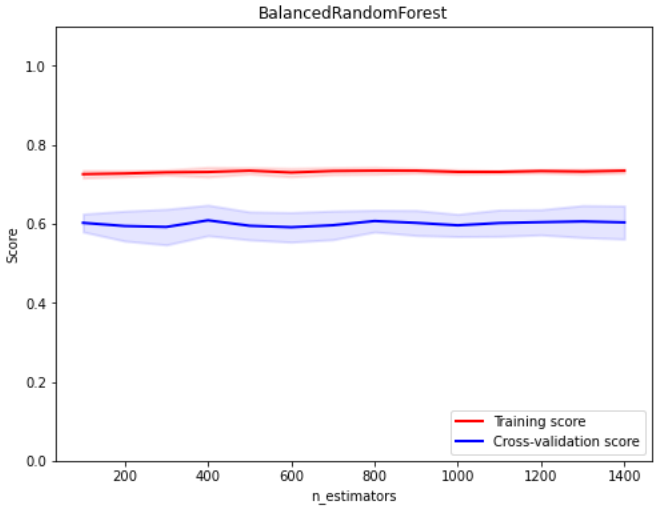


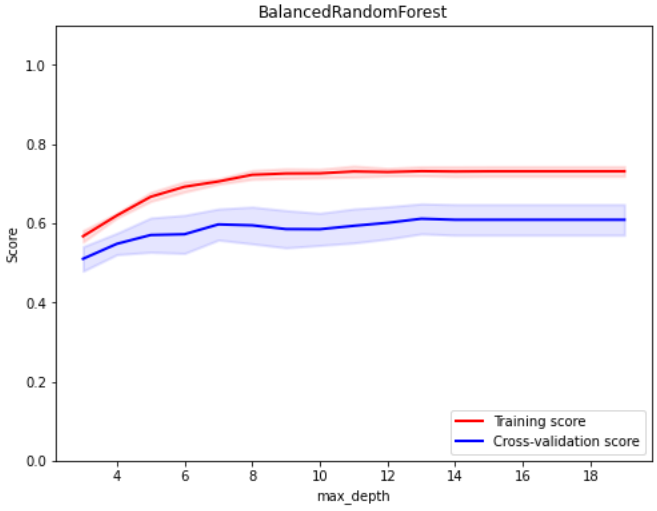
Figure 3: Accuracy and Varying Parameters in LR

4.2Balanced Random Forest

The major hyperparameters to tune are n_estimators and max_features. I tune 15 choices of n_estimators from 100 to 1500, max_features between “sqrt” and “log2”. I also tune max_depth from 3 to 20, 'min_samples_split,' and 'min_samples_leaf' from 1 to 5. Finally, I tune “bootstrap” from True or False. Based on step-by-step

hyperparameter tuning, I get the following functional relation graph between predictive accuracy and varying parameters. The following graph shows that the score reaches the highest when n_estimators = 400. The score goes high when max_depth hits 9. The optimal 'min_samples_split' and 'min_samples_leaf' are 2 and 1, respectively.





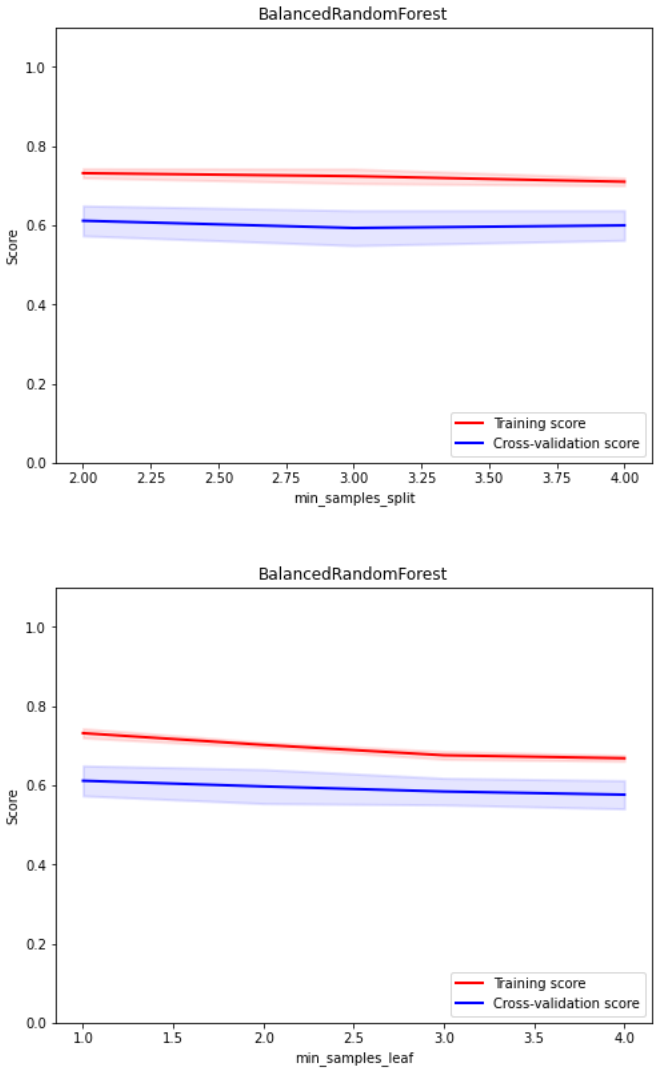


Figure 4: Accuracy and Varying Parameters in BRF

4.3 Balanced Bagging Classifier

For balanced bagging classifier, I tune the n_estimators, and 'max_depth', and 'learning rate' for the base estimator. The tuning process is the same as above, and following graphs about the changing parameter and scores tells the model is optimized when the number of estimator equals 400, then max_depth equals 9, and the learning rate equals 0.28.

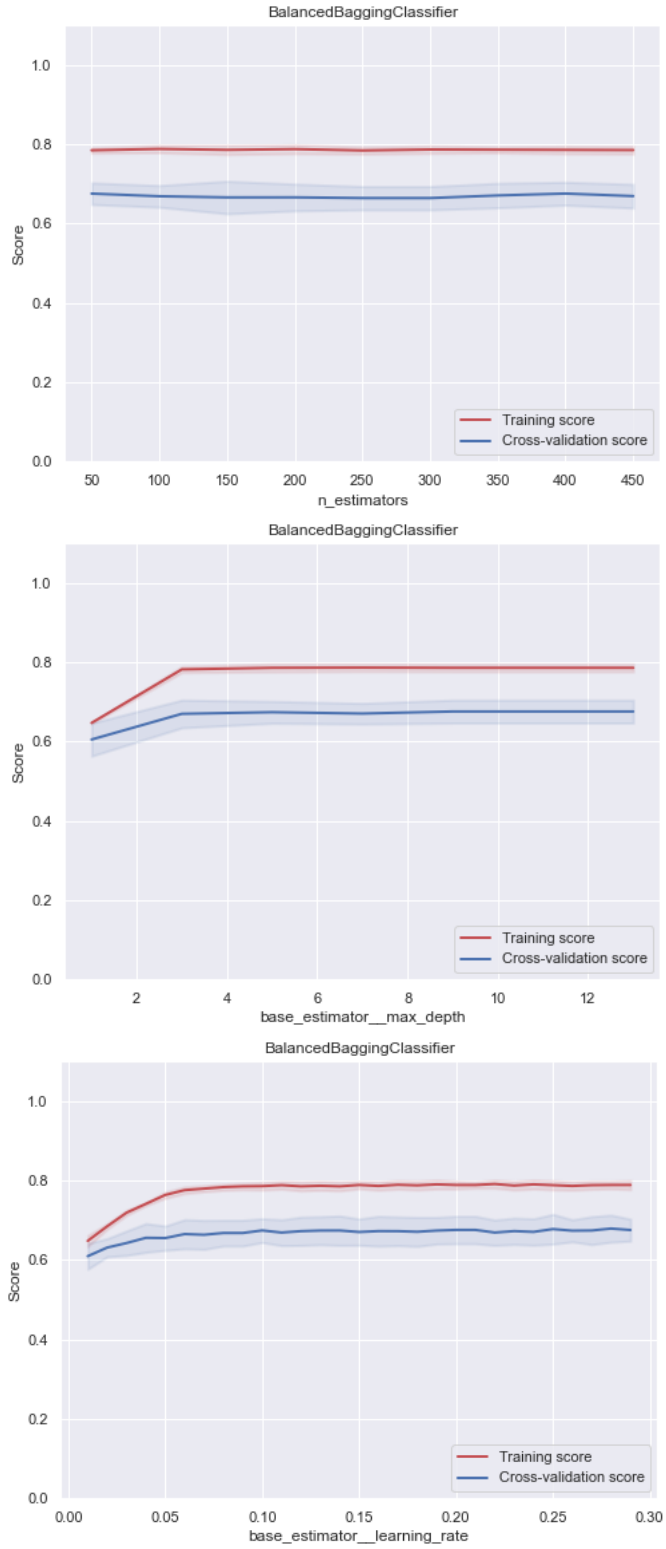


Figure 5: Accuracy and Varying Parameters in BBC

5 Data Splits

I don't split the dataset into a train set and validation set because I use cross-validation to train and validate each model. This cross-validation strategy provides a more comprehensive knowledge of model performance throughout the whole dataset rather than simply a single train/test split. StratifiedKFold() is applied to handle the imbalanced data, and the default number of folds is 5. It can help me make the most use of and not waste the train data.

6 Results

Table 2 summarizes the best F1 score for four models from either local or Kaggle. The local score is based on the cross-validation in grid searches, while the Kaggle result is offered when uploading the predicted results.

Table 1 Best Results on four models

Model	Train Time	Local F1 score	Kaggle Public F1 score	Kaggle Private F1 Score
Logistic Regression with Balanced Class Weight (LR)	0.03s	0.731	0.634	0.823
Balanced Random Forest (BRF)	3.15s	0.602	0.516	0.493
Balanced Bagging Classifier (BBC)	30.33s	0.679	0.6	0.697
Voting Classifier (LR+BBC)	20.36s	0.733	0.652	0.75

I selected the voting classifier that combines Logistic Regression with Balanced Class Weight and Balanced Bagging Classifier to submit in Kaggle since it provides the best F1 score in local cross-validation and public leaderboard. The voting classifier is more robust because it combines the advantages of each model. The voting classifier gets 0.75 on the private test set. More interestingly,

the logistic regression, which offers my second-highest F1 score, achieve 0.823 on the private set, ranking top 5 in the private leaderboard if I had chosen it!

References

[1] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[2] Agusta, Z. P. (2019). Modified balanced random forest for improving imbalanced data prediction. International Journal of Advances in Intelligent Informatics, 5(1), 58-65.

[3] Hido, S., Kashima, H., & Takahashi, Y. (2009). Roughly balanced bagging for imbalanced data. Statistical Analysis and Data Mining: The ASA Data Science Journal, 2(5-6), 412-426.

[4] Ruta, D., & Gabrys, B. (2005). Classifier selection for majority voting. Information fusion, 6(1), 63-81.

[5] Matt. Validation Curve Plot from GridSearchCV Results. Retrieved from https://matthewbilyeu.com/blog/2019-02-05/validation-curve-plot-from-gridsearchcv-results