Data-oriented Project: Project Proposal (Phase 1)

Chi Heng Jeffrey Hui. Author

School of Information Technology and Electrical Engineering

The University of Queensland, Qld., 4072, Australia

1. Pre-processing techniques

In the pre-processing step in phase 2, I will apply imputation, normalization, and outlier detection.

After observing the provided dataset, I can see there are many NaN values in the features, which means the dataset is incomplete, so we need to perform imputation. We can use SimpleImputer, IterativeImputer, and KNNImputer for imputation. SimpleImputer only accounts one feature, while IterativeImputer and KNNImputer account multiple features, which is useful when there are correlations between the features. As we don't know if the features are correlated or not, I decided to perform all three imputers. To understand which imputation methods works best, I will apply the processed datasets on different classification techniques. By using cross-validation, it will iterate k times, and we can get the means of the iterations. Afterwards, we will know how well the processed datasets perform with different models, which can help us to decide which is the best imputation method.

When observing the dataset, I can see feature values has different range. In example, column 6 has a maximum value of 4.49 and minimum value of -2.60 while column 3 has maximum value of 128.36 and minimum value of -99.48. Apparently, there are different degree of magnitudes on the features, so we need to apply normalization. Based on the distribution, we can choose max-min normalization or standardization. As the dataset is high dimensional, which is hard to visualize the dataset. Instead of guessing the distribution and deciding the method, I decided to perform both normalization method. To understand which method is better, I will apply the normalized datasets to different model. By using cross-validation, we can see which perform better on the models, so that we can pick the most appropriate normalization method.

Outlier is unnecessary, and may affect the classification results, so I also decided to apply outlier detection to see if there are outliers in the dataset. In scikit-learn, we may apply isolation forest or local outlier factor (LOF). Local outlier factor (LOF) doesn't work well high dimensional data. As there are 116 features in the dataset, it is a high dimensional dataset, so we won't apply LOF as our outlier detection method. Instead, we will apply isolation forest as the method to detect outliers. If we discover any outliers after applying the isolation forest, we will remove those instances.

To understand what are the best approaches to preprocess our dataset, we need to test different combinations with different classifiers, and we can average the score for each combination, which can help us decide which is the best combination. The tree diagram below in the last page shows all the possible combinations of the preprocessing techniques.

Base on the tree diagram , it shows there are 6 combinations of preprocessing techniques. We need to preprocess the dataset by each combination and apply them to the classifiers. By using cross-validation, we will get how each combination performs with different classifiers. Finally, I will calculate the average scores of how each combination perform with different classifiers. For instance, if combination A (Isolation forest, Max-Min, SimpleImputer) get 0.88 on decision tree, 0.9 on random forest ,0.95 on KNN and 0.85 on naïve bayes, the average score for combination A is 0.895. We will perform the same calculations on other combinations, the combination with the highest score is the selected preprocessing techniques for this project. Each combination will work with decision tree, random forest, KNN and naïve bayes. We don't perform hyperparameter tuning in this stage, since we are only looking for the best combination of preprocessing techniques, but not the final prediction model. When applying all the combinations to each classification model, we need to ensure the parameters values don't change, so that if the score change, we can account the changes is caused by the combinations. For example, dataset one is processed by combination A, and dataset two is processed by combination B. We want to apply both datasets to a decision tree model. In this case, the hyperparameters values of the decision tree model won't change, so the combination is the only variable that can affect the score.

2. Applying classification techniques

Once we picked the best combination of preprocessing techniques, and have our dataset processed, I will work on the models. In this stage, I will implement decision tree, random forest, k-nearest neighbor, naïve bayes first. Afterwards, I will ensemble the classifiers and compare its scores with other models.

To implement the decision tree classifiers, we need to import the decision tree classifier and cross_val_score. Afterwards, we need to create a decision tree object. To create a decision tree object, we need to figure out what are the appropriate hyperparameter values. After reading the documentation from scikit-learn, I realize most hyperparameter are either unnecessary, or its default already provide a good value. In this case, I decided not doing hyperparameter tuning for the decision tree model. Following, we can use the cross_val_score function. We need to pass the classifier, training data, scoring metric and cv value to the function. For every iteration of CV, we will get a score, and we will calculate the average by the end of the iteration. To find out the cv value that give the best score, we need to run cross_val_score with different cv values. I will test some common cv values, for instance, 5,10, and 30. If required, I can continue testing different cv values until it gives a satisfying score. The cv value that gives the best score will be selected to construct our decision tree model.

To implement the random forest classifiers, we need to import RandowForestClassifier. We need to create a random forest object. To create an object, we need to do hyperparameter tuning. After reading the scikit-learn documentation, I realize the hyperparameters are very similar with the decision tree, but we have n_estimators in random forest, which doesn't exist in decision tree. We will tune the n_estimators and max_features. The default value in the documentation of n_estimators is 100. In this case, I will try values ranging from 90-110. The max_features default is "auto". In this case, I will try values from 2 to 11.I will use GridSearchCV to implement the tuning, since it allows me to enter all testing value once only and it will also perform cross validation for every combination of hyperparameters, so that I don't need to run the same code multiple times. Through the result of the cross-validation, if I can see the mean score increase when max_feature and n_estimators increase, I may try larger n_estimators and max_features value until there are minimal difference. Once I get the best combination of n_estimators and max_features, I will also try different cv values to see which gives the best result, for instance, 5,10, and 30. The best combination of cv, n_estimators and max_features will be used as the hyperparameters to create the random forest model.

To implement the k-nearest neighbor, we need to import NearestNeighbors. We need to use different hyperparameters to create the KNeighborsClassifier object. After reading the documentation, I decided to do hyperparameter tunings on n_neighbors. The n_neighbors value should be odd number, so that a tie will not happen during the majority voting. I will apply 3,5,7,9,11 and 13 as n_neighbors value. To avoid running the same code multiple times, I will also use GridSearchCV. GridSearchCV performs cross-validation for each hyperparameter value, so every folder has chance of being testing data. The n_neighbors value that has the highest score of CV will be the best n_neighbors value. Afterwards, I will also change the cv value and see what cv value works best, for example,5,10, and 30. If required, I can test more cv values. The best combination of n_neighbors and cv will be used as the hyperparameters to create the k-nearest neighbor model.

To implement the naïve bayes, we need to consider different types of naïve bayes method. Multinomial NB is for discrete features, in example, word counts. As our dataset doesn't match this description, it is not applicable. Bernoulli Naïve Bayes may not work with our dataset as well, as it assumes to be binary features, but not all our features are binary. We can try Gaussian NB, Complement NB and Categorical NB as our naïve bayes model. To perform these naïve bayes model, we need to create an object first. I decided using the default hyperparameter values for all these NB models. I will use the cross_val_score function to perform cross-validation with each NB model. The NB model with the highest cross-validation score will be the model for naïve bayes. In addition, I will also try different values of cv, for instance, 5,10 and 30. If required, I will test more cv values for better model.

To implement the ensemble method, I will use the Voting Classifier in the ensemble methods. As the Voting Classifier will get the results by majority voting, we need to supply odd number classifiers to prevent tie. In this case, I will pick the best three classifiers I did from the previous steps. Also, I will apply cross validation during the process, so that every row of data can become a testing data, which will give us an unbiased score.

Finally, we need to compare the scores of the decision tree, random forest, KNN, naïve bayes, and the ensemble classifiers. The classifier with the highest score will be our final model of this assignment.

3. Evaluating the Model

To evaluate every model in this assignment, we need to perform cross-validation. By using cross validation, one folder will be the validation data, and other folders will be the training data. When each iteration is completed, the validation data will move to the next folder. Each iteration will give us a score, and we will calculate the average when the iteration is completed. Cross-validation gives us an overall score of how the model performs, so that it can prevent inaccuracy. For instance, if you KNN model give you 0.95 for the first iteration but get 0.85 and 0.75 in the remaining iterations. Your cross-validation score should be 0.85. If cross-validation doesn't perform, we may get 0.95 as our final score, which is not correct.

After analyzing the dataset carefully, I realized the dataset is imbalanced. There are 1500 rows of data in the dataset. For the target, we only get 161 rows of data that has a value of 1. And we have 1339 rows of data has a value of 0. As compared the numbers we can see one class has very low proportion in the data when comparing with another class. As accuracy may not work well with imbalanced data, so I decided to use F1 as a metric to evaluate the classification performance.

4. Timeline

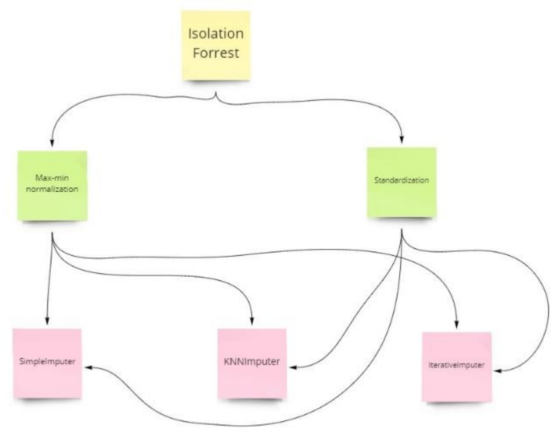
Milestone 1(17/9/2021-24/9/2021): In this milestone, I will focus on the preprocessing techniques. I need to test how the combinations of preprocessing techniques affect the overall performance of a model. By the end of this milestone, I can obtain the best combination to preprocess the project dataset.

Milestone 2(24/9/2021-1/10/2021): In this milestone, I will focus on the decision tree, and random forest. By using hyperparameter tuning techniques, I should get the best decision tree and random forest model by the end of this milestone.

Milestone 3(1/10/2021-8/10/2021): In this milestone, I will focus on KNN and naïve bayes. By using hyperparameter techniques, I should get the best KNN and naïve bayes model by the end of this milestone.

Milestone 4(8/10/2021-15/10/2021): In this milestone, I will focus on the ensemble classifier, model selection and the report. By the end of this milestone, I should complete the ensemble classifier, and I should be able to select the final model for the dataset. In addition, I should start working on the report.

Milestone 5(15/10/2021-29/10/2021): In this milestone, I need to focus on the project report. Once the report is completed, I should proofread the report. In addition, I also need to organize my codes and add appropriate comments to make it easy to follow and read.


Isolation Forrest
Max-min normalization
Standardization
SimpleImputer
KNNImputer
IterativeImputer
miro