IEE 520 Final Project

Javier Leon Mendez

Fall 2020

# Data Pre-processing

The data provided was initially explored as a pandas data-frame to get an idea of the dimensions of the matrix, missing values, range of the values in the features provided, and the class balance. Figure 1 shows a portion of the data-frame exploration. In it, it can be seen that some of the features have missing values. One way to handle missing values is to delete them from the dataset, but I want to be able to preserve the rows with missing values. Imputing is the process of filling the missing values with data that may be similar to the one that is missing. Scikit Learn provides an imputer method with several options to fill in missing values.

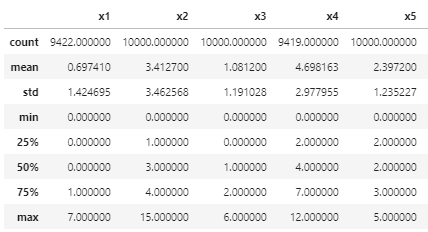


Figure 1 Data-frame description output showing missing values for features X1 and X4

Additionally, Figure 2 shows that for the variable features there is a difference in the order of magnitude of the average values in features X16 to X19. These differences can cause sensitivities in the way some algorithms are trained; therefore it is a good idea to scale the data before training the given algorithm.

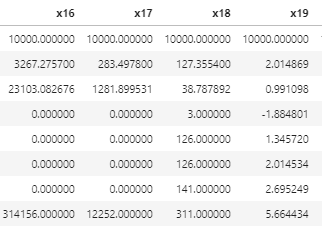


Figure 2 Dataframe description output showing orders of magnitude difference in the average values for variable features

## Class Imbalance

Over sampling was performed on this data to address the class imbalance. The class with the least examples, in this case Class 1, was randomly oversampled until the size matched the size of Class 0. The oversampled Class 1 was then merged back in with Class 0 to give a balanced data for training and validation.

# Methods and Evaluation

The selection criteria for the algorithm for this project was based on choosing a model that could handle missing data, that could handle both categorical and variable features, that would train fast and did not have many hyperparameters to tune. I decided to use Extreme Gradient Boosting (xgboost), since based on my research is one of the best implementations of boosted decision trees, an algorithm we covered in class. When training the first model I used the default parameters, the unbalanced data which I split into 80% for training and 20% for validation and obtained a Balanced Accuracy of 77.9%. After using the balanced data with the default parameters, the Balanced Accuracy improved to 91%. I also obtained the following Confusion Matrix:

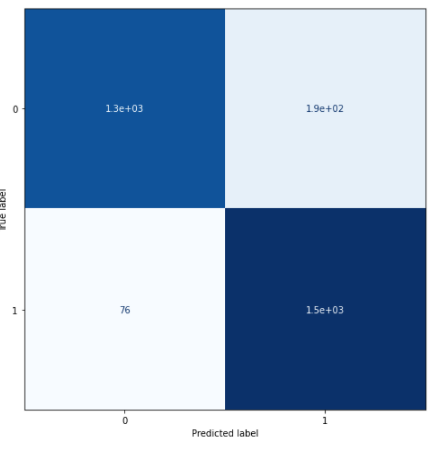


Figure 3 Confusion Matrix of the xgboost model with default parameters and balanced data

# Model Selection

For the final model I used all the data provided to train it. I also used scikit learn GridSearchCV to find the best hyperparameters for xgboost. The following code snippet details the parameters used to train the final model:

xgb\_final\_model = XGBClassifier(nthread=4,seed = 519)

parameters = {'n\_estimators': range(100, 1000, 100), 'learning\_rate': [0.1, 0.01, 0.05, 0.001] }

grid\_search = GridSearchCV( estimator=xgb\_final\_model, param\_grid=parameters, scoring = ' balanced\_accuracy’, n\_jobs = 4, cv = 5, verbose=True)

Grid search found that the best parameters to use for a model trained on all the data were number of estimators = 900, and learning rate of 0.1. As well the mean cross-validated balanced score improved to 93.5%. I then trained a model using those parameters on all the data and proceeded to predict the unlabeled data. See filename “IEE520BMI555Report2020JavierLeonMendez.csv” for unlabeled data predicitons.

# References

Chen, T., & Guestrin, C. XGBoost: A scalable tree boosting system. arXiv 2016. *arXiv preprint arXiv:1603.02754*.