CS434: Final Project Report

Evan Newman and Kevin Le

June 12th, 2019

1 Data preprocessing

We utilized the imbalanced-learn library which extends off of the sklearn library for two of the three algorthims (Decision Trees and Random Forest). We didn't use any data preprocessing for our Neural Network mainly for implementation issues. This library has many approaches in order to handle imbalanced data and after looking at the methods available we decided to use it's implementation of Synthetic Minority Oversampling Technique (SMOTE). The decision for this over-sampling technique was made because for both the Decision Tree's and the Random Forest's algorithm just copying the minority class wouldn't do much to change the Decision Tree or Random Forest's training considering it's already taking into account that training sample. Since SMOTE creates new samples of the minority class we predicted that the algorithms would pick better decisions for splitting the data. The only concern we had about using this technique is that the values in our training set are representations of biology which has set rules, but considering we didn't see much change when removing SMOTE in our accuracy we decided to keep it. One note is that imblanced-learn has a BorderlineSMOTE() option which focuses on samples near of the border of the optimal decision function and will generate samples in the opposite direction of the nearest neighbors class. Considering regular SMOTE might give us odd results we decided to use this specific implementation of the SMOTE technique.

2 Learning algorithms

2.1 Algorithms/methods explored

1. Neural Networks: We believed that the neural networks could potentially correct for the imbalance in data. It also has a lot of options available such as momentum, dropout and learning rate that could adjust to optimize results.

2. Decision Trees: Searching online, several articles have stated that decision trees are inherently better for handling imbalanced data because it forces itself to take into account both classes for splitting so it seemed like a good pick as a base algorithm to compare to the rest of.

3. Balanced Random Forest: While searching for appropriate models, sklearn had an implementation of Random Forest. However, imblanced-learn's

library had an extension of this called BalancedRandomForestClassifier() which looked to fit our needs. Random Forest seemed like a good option since it's a ensemble method which is something that really helps for imbalanced data.

4. Naive Bayes: We did explore this option, though not very in-depth, only simple for the reason that a classmate told us they got good results with the Naive Bayes algorithm.

5. Support Vector Machines: This method was also explored from the reasoning that SVM utilize the handling of class penalties which, from what we gathered online, is said to be a good method for dealing with imbalanced data.

2.2 Final models

1. Neural Networks: A MLP neural network was trained with momentum = .5, dropout = .2, and learning rate = .0001. The data was formatted to look like a (1 X 103) or (1 X 1053) image which was fed into the model as data. There was only one hidden layer for our neural network. For 103 features, there were 103 nodes initially which reduced to 50 nodes, and those 50 nodes reduced down to 2. For 1053 features, there were 1053 initial nodes which reduced down to 500 nodes, and then from 500 nodes reduced down to 2 nodes.

2. Decision Tree: This algorithm utilized sklearn.tree libraries DecisionTreeClassifier() function. The paramaters passed into the DecisionTreeClassifier was critierion='entropy' and max_depth=5. Once finished the sklearn.fit() function was applied to the training data. After the data had finished, to get the prediction, sklearn.predict_proba() was called which passed into it was the test data. The final step was just grabbing the second elements of the array that predict_proba() returns which is the probability of the positive class.

3. Balanced Random Forest: This algorithm utilized the imbalanced-learn.ensemble libraries function BalancedRandomForestClassifier. The parameters passed into it were n_estimators=100 and max_depth=5. After the data had finished, to get the prediction, sklearn.predict_proba() was called which passed into it was the test data. The final step was just grabbing the second elements of the array that predict_proba() returns which is the probability of the positive class.

3 Parameter Tuning and Model Selection

3.1 Parameter Tuning

Parameters tuned for our DT and RF models were the different types of SMOTE implementations, that being imbalanced-learn's normal SMOTE implementation and it's modified BorderlineSMOTE implementation. In addition to our sampling we tweaked with different max-depths for our Decision Tree and Random Forest algorithms. We decided on 5 for our max-depth because

the algorithms would over-fit around a depth of 10 and picking 5 seemed like a nice in between, also we didn't see any significant changes in our accuracy when picking max-depths of 4, 5, and 6.

Another tuning we went through was how big our test_size was for our model-selection. We went for one-third of our training data as a validation set just to see how well our algorithm was performing before using our entire training data to train on to predict on our actual test set.

For neural networks, the parameters tuned were momentum, learning rate, epochs, and dropout. It should be noted that there are more parameters that could be tuned such as number of hidden layers and weight decay.

3.2 Model selection

As we had a decent amount of data points, we used hold out validation and we used a random sampler to select one third of the training data for our validation while the rest would be used for training. Also mentioned is our use of SMOTE(), a method for oversampling using the entirety of the data set. These two methods together, we hoped would deal with imbalanced data. It should be noted that this method was only applied to our two best performing models. The criteria we used for selection of our models was based on how well it classifies the validation set and how it performs when their output files are fed in sklearn's roc_auc_score() function.

Using this method of validation, we managed to get the highest accuracy with, from highest to lowest, 1) balanced random forest, 2) decision trees, and 3) neural networks thus prompting our decision to go with these three models.

4 Results

Based of sklearn.roc_auc_score() the accuracy from our models for the 103 features data are thus:

Balanced Random Forest: $\sim 92\%$
Decision Tree: $\sim 88\%$
Neural Network: $\sim 70\%$