Group Members:

Vasu Chaudhary – vsc3
Sanket Sinha - ssinha27
Roshini Seshadri – roshini3
Kaggle Id: rsv

Introduction:

The goal of the project is to prognosticate the risk level (response variable) in providing insurance to new clients based on the historical data of various clients. We implemented a random forest classification algorithm to classify this. Though we tried to accommodate this classification problem using a decision tree with pre-pruning techniques, we believe that it over fits the data. A random forest algorithm performs bagging to combat this, it is robust to overfitting and the effects of noise in the data.

Detailed Project Description

Following are the various stages:

1. Preprocessing/ Data Cleaning

   In this stage, we separated the different types of features in the dataset- Object (String), categorical, and numeric. For string features, we converted it to categorical variables and implemented heuristics like most common. Numeric features were handled by replacing the null values with the mean. Apart from this, the numeric values were subjected to binning with a bin value of 2, which returns a pandas series of type categorical. This step is necessary to convert the wide range of continuous values into categorical ranges.
   We removed the Id feature from the training and the testing set because it was a unique column and hence, would result in an extravagant number of splits, consequently leading to overfitting of data.

2. Random Forest:

   As a first step, we tried out a range of values for setting the number of trees in the random forest. We noticed that the performance, both in terms of speed and accuracy, was the best when the number of trees were 25. As we increase the number of trees, the accuracy increases. To further improve the speed, we implemented multi-processing using the multiprocessing library in python. Multi-processing was a logical solution to improve the speed because it runs each tree as a separate process and the number of processes executed at a time to the number of cores which is like sklearn’s implementation of random forest [n_jobs = -1].

3. Bootstrap sampling:

We are making use of bootstrap sampling with 95% of features. Class imbalance problem was addressed in this section, it is evident from the 'Response' feature that there's an uneven distribution of the classes. To combat this, 2000 records from each class were selected. However, not every class has that many records, hence for those classes, the replace parameter was set to true when sampling was performed. This ensured that all the classes were balanced. This sampled data was then passed to the decision tree class.

4. Decision Tree:

   This is the crux of the algorithm where the actual tree generation happens. We used the gini criterion as the attribute selection measure. The filtered data is recursively passed to compute the gini index and build the tree. The predict function traverses the tree and returns the prediction array from which mode is computed to decide the final response of the tree.

   When we initially tried classifying with just a decision tree (without pruning), though the training time was not much, it did not perform well with the testing data. Random decision trees by Tim Kan Ham elucidates the fact that using multiple trees in addition to increasing accuracy using bagging, it also handles the overfitting.

5. Pre-pruning techniques:

   We have employed techniques like altering the max-depth and min-samples-split to avoid overfitting the data. Min samples split is the minimum number of samples that is required to split the node and max depth, this feature stops the tree from proceeding till all the leaves are pure. These parameters considerably increased the performance of the decision tree with an execution time of approximately ***1.5 minutes per tree***.

   When min samples split is set to a low value, it would make the tree over fit and learn from the noisy data too. We set it to 142 which gave a good execution time and accuracy, that is it doesn't prematurely classify the noisy data.

   Max depth feature is used to give an equal chance to all the features to become a decision node, however, it should not be too much that we split on every value that exists in a feature. A max depth of 14 gave us an optimum result both in terms of accuracy and speed.

Results

Random forest:
n_estimators:25
Max depth: 14
Min samples split: 142

Bagging ratio: 0.95
Score: 0.48268

Decision Tree:
Max depth: 14
Min samples split: 142
Score: 0.45015

References:

[1] http://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html

[2] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[3] Tin Kam Ho, "Random decision forests," *Proceedings of 3rd International Conference on Document Analysis and Recognition*, Montreal, Que., 1995, pp. 278-282 vol.1. doi: 10.1109/ICDAR.1995.598994

[4] https://docs.python.org/2/library/multiprocessing.html