**Introduction**

Class imbalance remains an important issue in the area of machine learning. It has been shown that a classification model with imbalanced classes will perform poorly with respect to predictability. There are many techniques that can be used to more evenly balance the data. These include oversampling the minority class, under sampling the majority class, and some combination of both. The SMOTE function from the DMwR package in R utilizes a combination of both oversampling and undersampling. However, it is unclear what combination of either is best for prediction. Here, we aim to test eight types of over and undersampling techniques to evaluate which combination creates the most predictive model.

**Methods**

Publicly available topologically associating domain data was obtained from GEO with accession GSE53525. The domain data was in the form of two-column genomic coordinates from the GM12878, with each coordinate representing a TAD boundary. The coordinates were concatenated into a long vector and were sorted. The genome was then binned into a series of 1kb intervals (500 bases on either side of the boundary point) and an indicator vector Y was created based on whether there was a TAD boundary in the 1kb interval (Y=1) or not (Y=0). There were 16,700 TAD boundaries out of 2766314 possible 1kb intervals. Therefore, the class probabilities were given as: 0.994 for Y=0 and 0.006 for Y=1.

Various genomic annotations were obtained from the GM12878 cell line of the ENCODE Project. These annotations made up the feature space **X**, including p predictor variables ({X₁,...,X­p}). Two types of predictors were created for each annotation. First, a binary variable was created denoting whether or not the 1kb bin overlapped with the annotation or not. Second, a continuous variable was calculated for each annotation as the distance to the nearest coordinate from the 1kb bin. The total feature space consisted of 103 predictors.

In order to create balanced classes, the SMOTE command from the DMwR package was implemented. SMOTE stands for Synthetic Minority Over-sampling TEchnique. Both majority undersampling and minority oversampling is used to create balanced data. The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors (here, k=5). Then, the majority class is under sampled by taking a random sample that matches some pre-specified percentage of the updated minority class. For example, consider a data set with 700 observations, where there are 35 minority classes and 665 majority classes. Now, when using SMOTE, consider a 200 percent over sample of the minority class and a 100 percent under sample of the majority class. This means that two synthetic observations will be created for every minority class (35+70=105) and 1 majority class will be randomly chosen for every new minority class created (70), resulting in a classification of 70/105 (majority/minority). Now consider, a 200/200 combination. The new minority class is still 105 (70 new synthetic observations created), but now twice as many majority classes are randomly chosen (70\*2=140), resulting in a 140/105 classification. It should be noted that using a combination of 100/200 will always result in perfectly balanced classes.

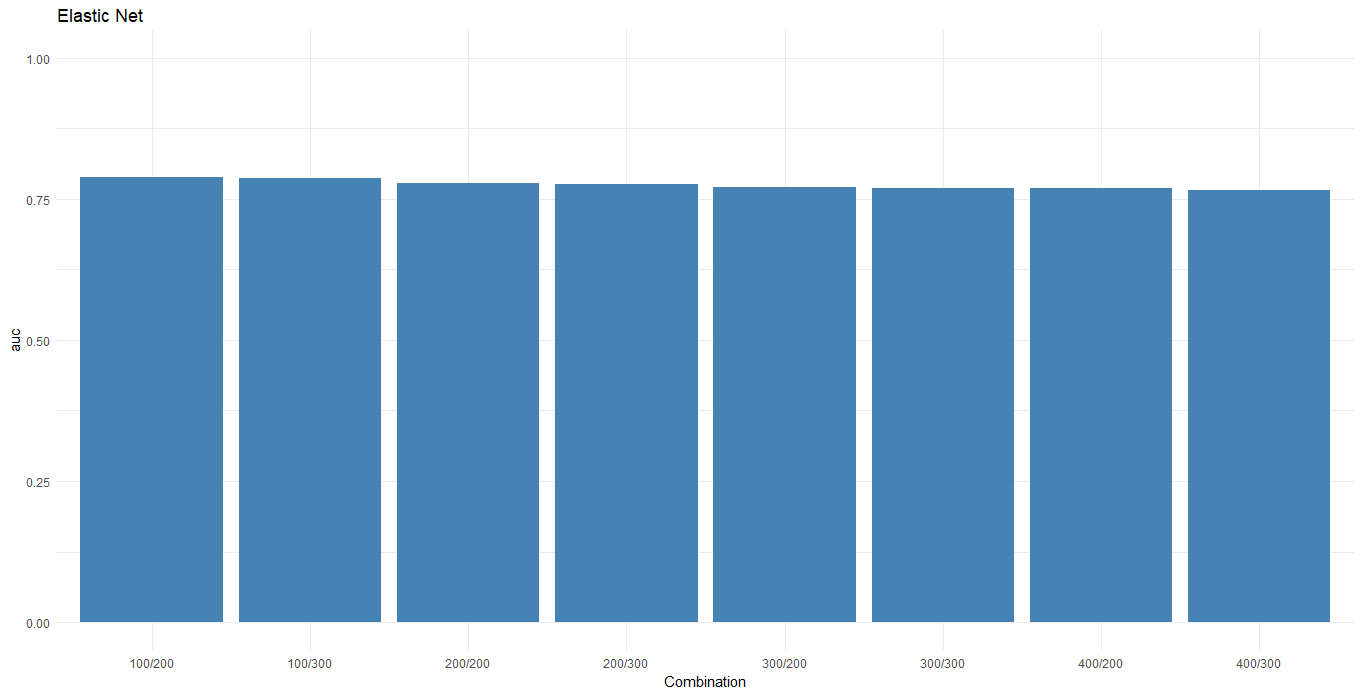
Three models were implemented, an elastic net, random forest, and gradient boosting machine. Each model used 5 fold cross validation, repeated 3 times in order to tune for the respective model parameters. A total of eight different combinations of percent over and undersampling were used which included: 100/200, 200/200, 300/200, 400/200, 100/300, 200/300, 300/300, 400/300. Here, the 100/200 combination creates a completely balanced dataset, while the others create an unbalanced dataset with the “No” class as the majority to reflect the original data. Model performance was evaluated using ROC curves and corresponding AUCs. Feature importance plots were also provided. All analysis was perfomed in R version 3.4.2.

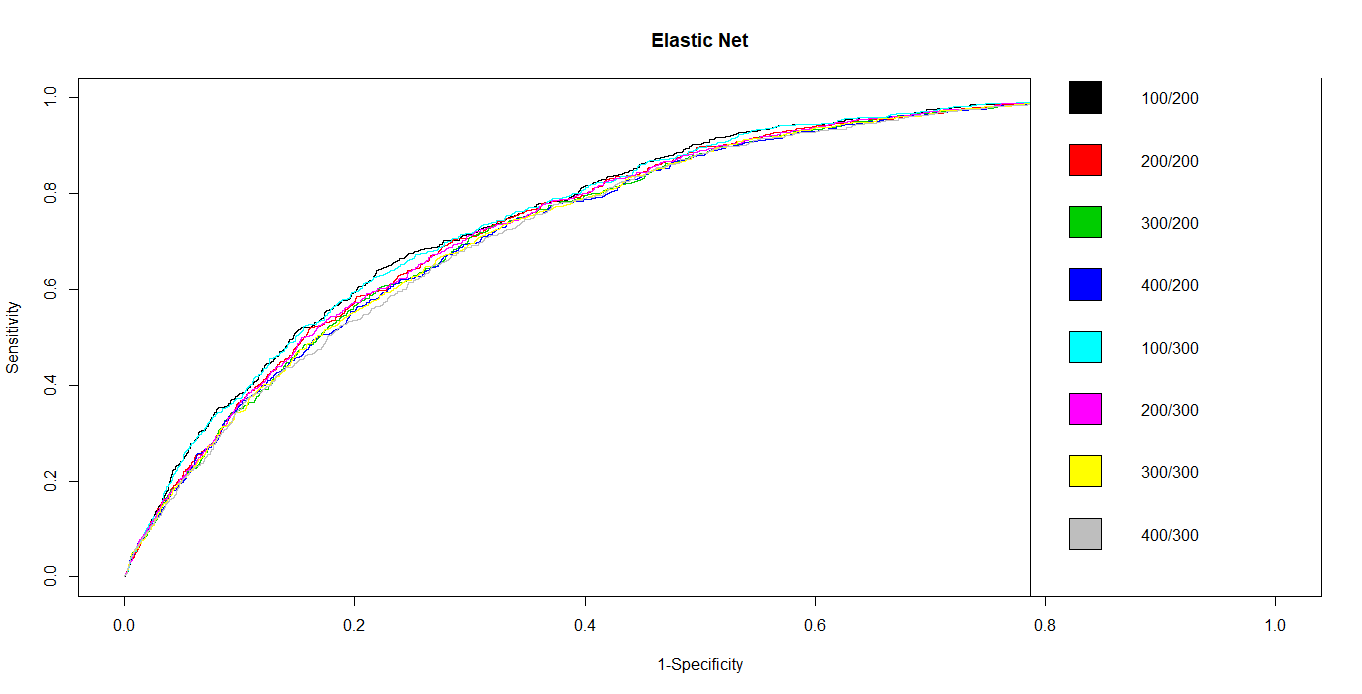
**Results**

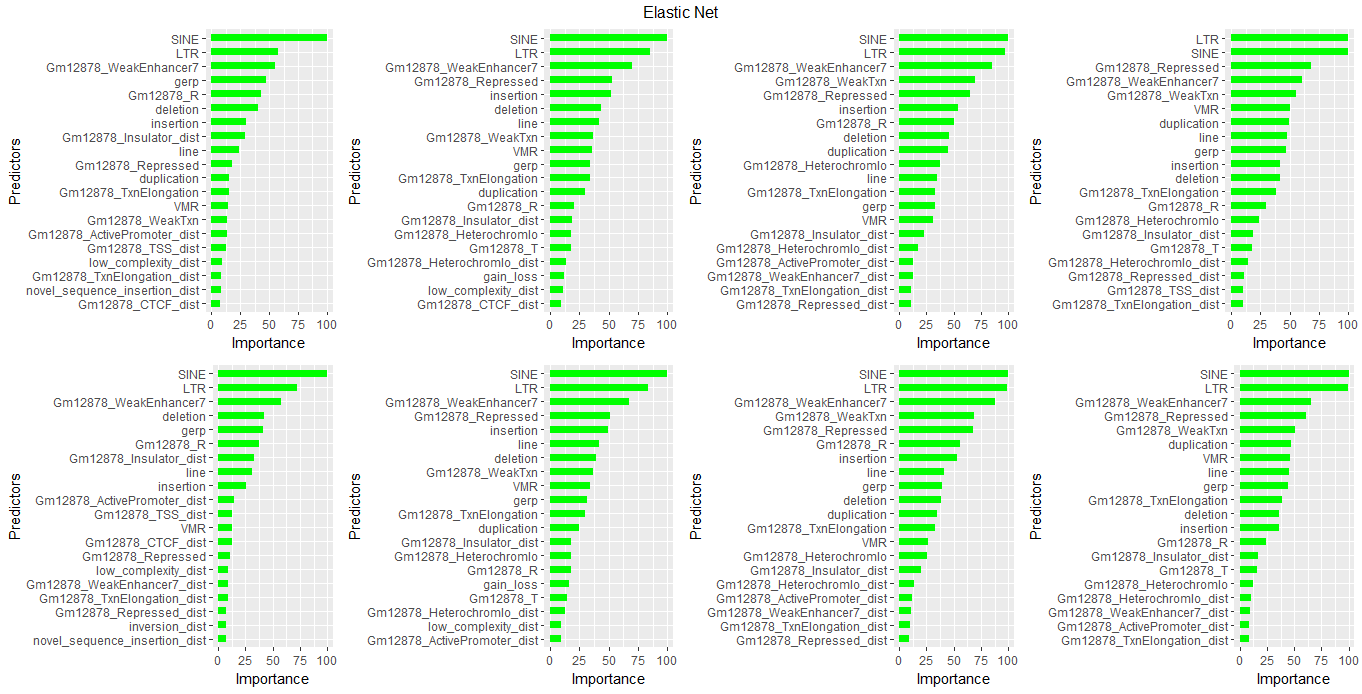
|  |  |  |  |
| --- | --- | --- | --- |
| **SMOTE Combination** | **Number of Observations** | **Class Balance** | |
| **No** | **Yes** |
| 100/200 | 4564 | 2282 | 2282 |
| 200/200 | 7987 | 4564 | 3423 |
| 300/200 | 11410 | 6846 | 4564 |
| 400/200 | 14833 | 9128 | 5705 |
| 100/300 | 5705 | 3423 | 2282 |
| 200/300 | 10269 | 6846 | 3423 |
| 300/300 | 14833 | 10269 | 4564 |
| 400/300 | 19397 | 13692 | 5705 |

**Table 1.**

|  |  |
| --- | --- |
| **Elastic Net Model Performance** | |
| **SMOTE Combination** | **AUC** |
| 100/200 | 0.788 |
| 100/300 | 0.787 |
| 200/200 | 0.777 |
| 200/300 | 0.776 |
| 300/200 | 0.771 |
| 300/300 | 0.770 |
| 400/200 | 0.769 |
| 400/300 | 0.766 |

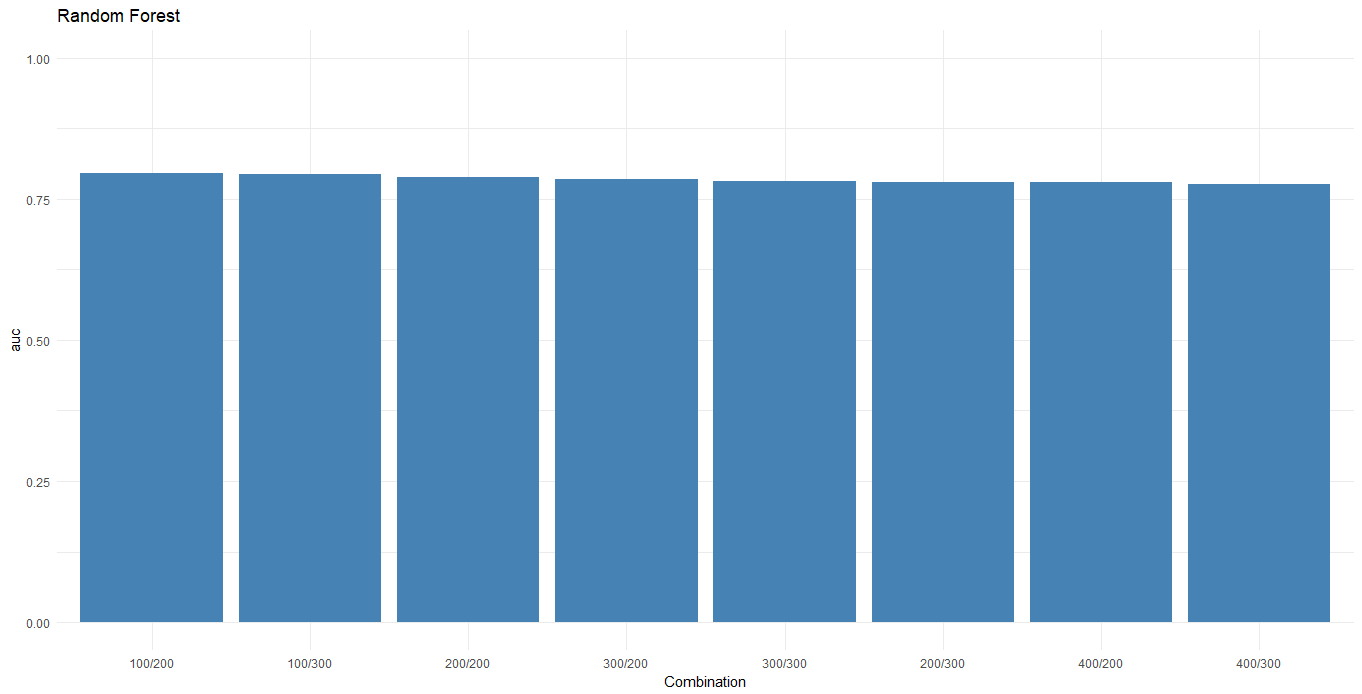


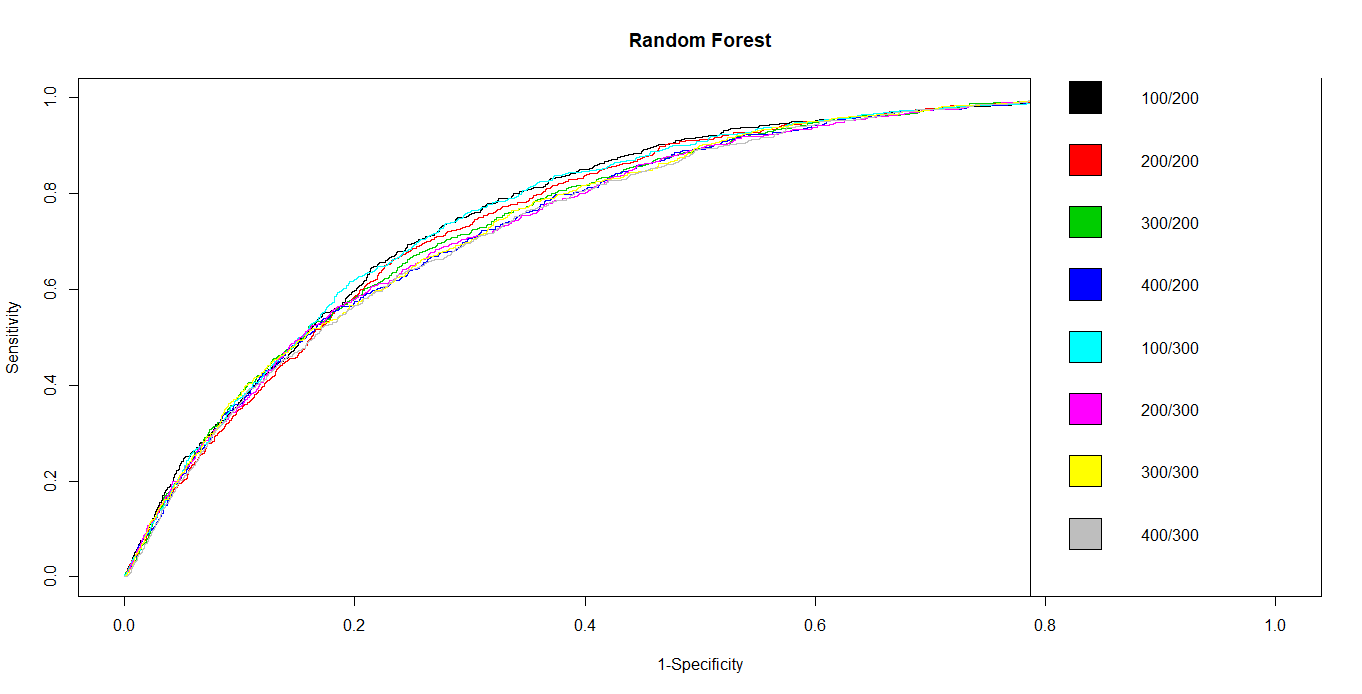


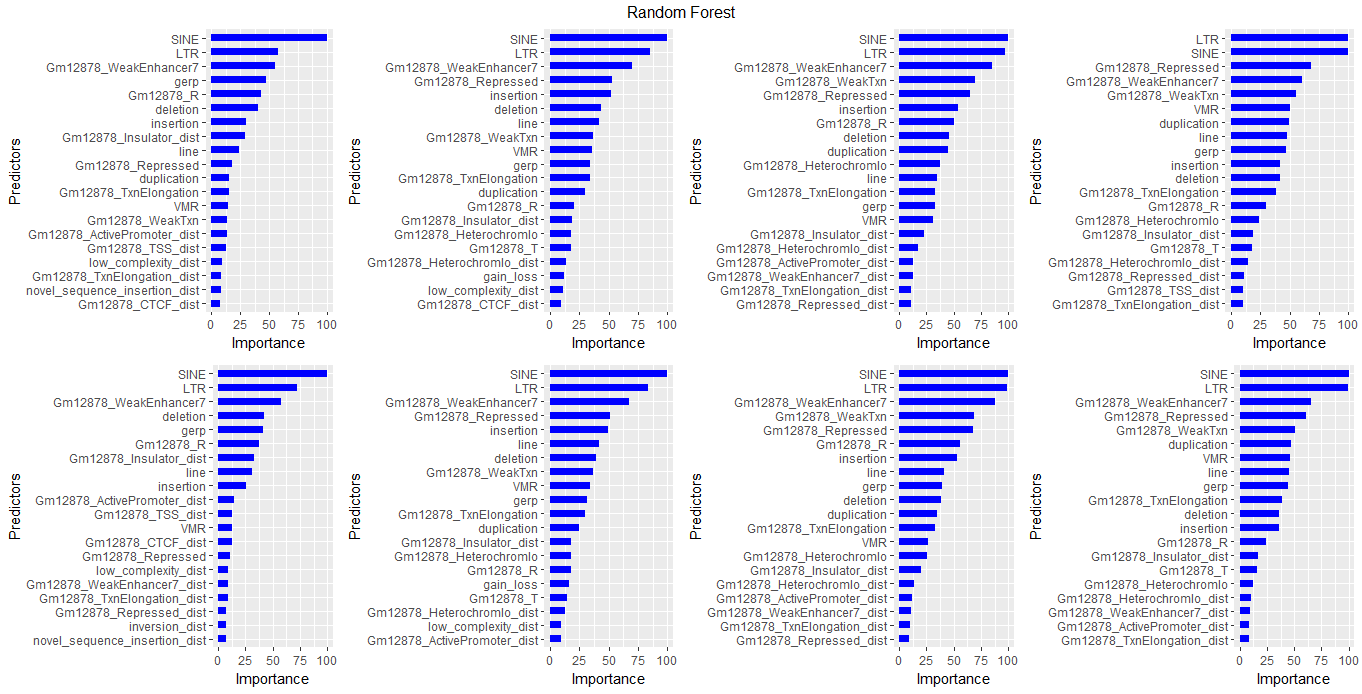


**Figure 1.**

|  |  |
| --- | --- |
| **Random Forest Model Performance** | |
| **SMOTE Combination** | **AUC** |
| 100/200 | 0.796 |
| 100/300 | 0.794 |
| 200/200 | 0.788 |
| 300/200 | 0.785 |
| 300/300 | 0.782 |
| 200/300 | 0.780 |
| 400/200 | 0.780 |
| 400/300 | 0.778 |

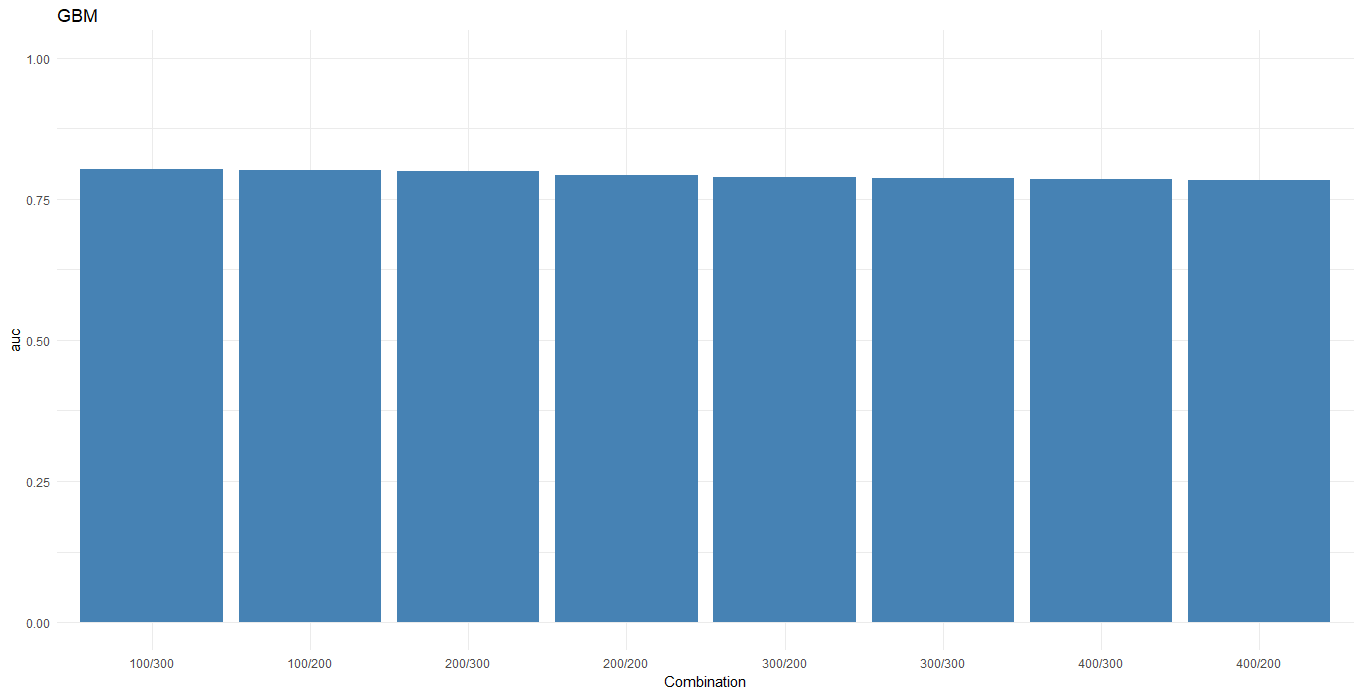


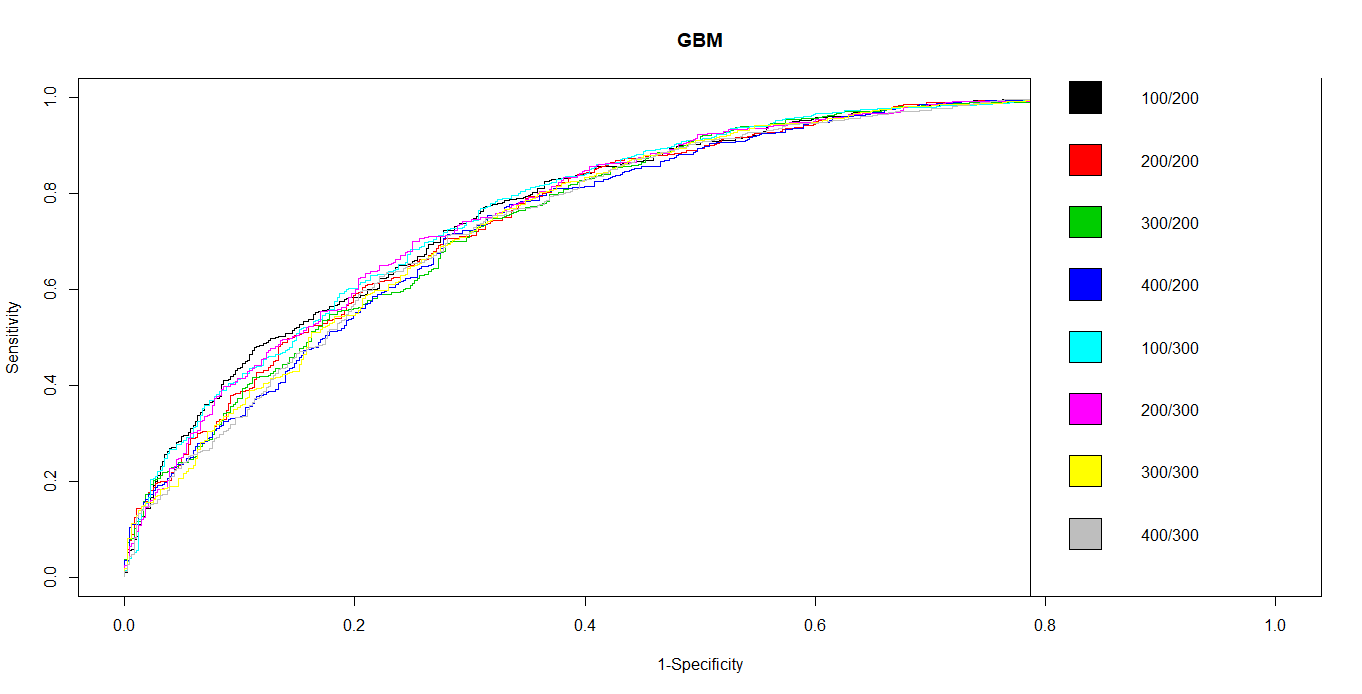


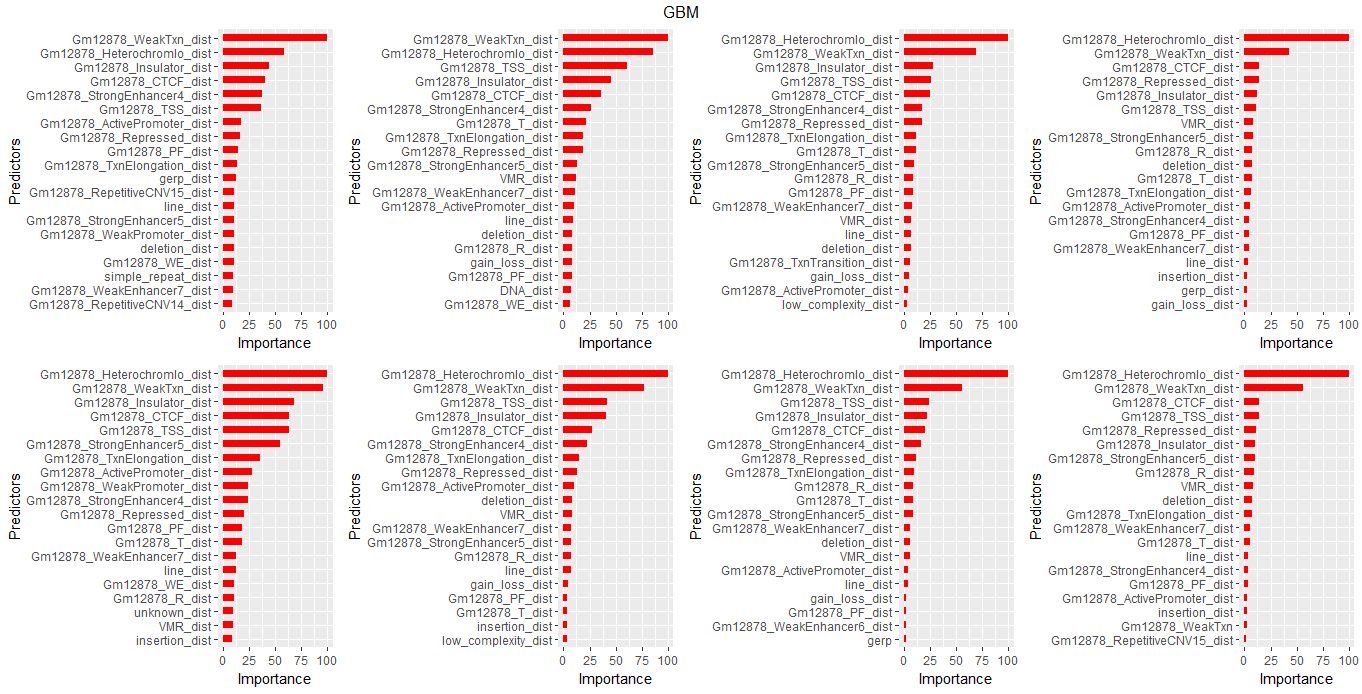


**Figure 2.**

|  |  |
| --- | --- |
| **GBM Model Performance** | |
| **SMOTE Combination** | **AUC** |
| 100/300 | 0.803 |
| 100/200 | 0.802 |
| 200/300 | 0.800 |
| 200/200 | 0.792 |
| 300/200 | 0.788 |
| 300/300 | 0.787 |
| 400/300 | 0.784 |
| 400/200 | 0.783 |







**Figure 3.**