Machine Learning Project

Latera Tesfaye Olana

08 March 2023

# Abstract

The forthcoming comprehensive report endeavors to utilize various machine learning models to forecast patients with Alzheimer’s disease, in contrast to healthy elderly individuals, through an analysis of cerebral cortex thickness measurements. This report will present a range of models, from simple linear models to intricate non-linear models.

# Introduction

The rationale behind employing multiple models instead of relying on a single model lies in the fact that each machine learning model exhibits a distinct set of strengths and weaknesses, and there is no one model that universally outperforms others. By assessing various models, we circumvent potential biases that may arise from selecting a single model, thereby augmenting our confidence in the accuracy and generalizability of our findings. Furthermore, we mitigate the risk of data snooping and concentrate our efforts on the development of diverse models.

# The Data

The data has zero missing values. In addition, it has 97 C outcome (hereafter, refereed to as no disease) observations and 303 AC (hereafter, referred to as diseased) observations. This shows the data is significantly unbalanced (Figure A1, in the supplementary section). The data has 360 predictors (p=360). Overall, the data has 400 observations. The data is divided into two, training and test with each containing 280 and 120, respectively.

Figure A2 in the data supplementary section, shows the scatter plot of the outcome for *P* values from 10 to 13. It doesn’t seem like there clearly defined boundary for these predictors. However, if we were to reduce all predictors into two predictors (using PCA), as shown in figure A3 of the data supplementary section we can see that there is a clear distinction between distribution of the outcome values. This might imply linear simpler models might describe the data well. For non-linear relationship, it may be more appropriate to use a non-linear model, such as a decision tree, random forest, gradient boosting, or a neural network. These types of models can capture more complex and nuanced relationships between predictors and the target variable, as opposed to linear models.

An alternative approach is to employ a K-nearest neighbors (KNN) model, which does not impose any predetermined mathematical function for relating the predictors to the target variable. Instead, this technique uses the closeness of data points to predict outcomes.

Figure A4 of the supplementary material shows the correlation between each predictor. Even though, it difficult to see the correlation between individual predictors, we can generally see there is smaller correlation.

# Models

## Simple Tree (Decision tree - unpruned)

We begin our journey with a basic tree structure. Figure B1 and B2 in the supplementary section of simple tree depict the structure of full-grown tree on the entire data and train data, respectively. Tree-based algorithms are a type of non-parametric model that divide the feature space into smaller regions with similar response values through binary recursive partitioning. Partitioning in classification problems typically maximizes the reduction in cross-entropy or the Gini index.

Table 1. Confusion matrix and accuracy plot for decision tree

| Train | Control | Alizahmer |
| --- | --- | --- |
| Control | 58 | 7 |
| Alizahmer | 10 | 206 |

|  | Accuracy train | Accuracy test |
| --- | --- | --- |
| Accuracy | 0.93950 | 0.80672 |
| Sensitivity | 0.96714 | 0.86667 |
| Specificity | 0.85294 | 0.62069 |
| Pos Pred Value | 0.95370 | 0.87640 |
| Neg Pred Value | 0.89231 | 0.60000 |
| Prevalence | 0.75801 | 0.75630 |
| Detection Rate | 0.73310 | 0.65546 |

Confusion matrix for test data

| Test | Control | Alizahmer |
| --- | --- | --- |
| Control | 18 | 12 |
| Alizahmer | 11 | 78 |

As shown in the summary table 1 the train accuracy is 94%, whereas the test accuracy is 81%. To avoid over fitting the depth of the tree must be selected appropriately. To do that we will use pruning. In this exercise I used *rpart* package for this simple tree development. Figure B3 in the supplementary shows the complexity, cross-validation accuracy with different sizes of tree. *rpart* has internally, built *k-fold = 10* cross validations. Figure B3 in the supplementary shows tree size versus complexity parameter (*cp*) plot for this default value. Accordingly, *the* computed best value for cp based on the smallest error, is 0.0515.

## Simple Tree - pruned

We will find the optimal subtree by using a cost complexity parameter () that penalizes our objective function for the number of terminal nodes of the tree (*T*).

minimize {R(T ) + α|T |}

Where *R(T)* is training error of leaf nodes; *|T|* — The number of leaf nodes; and complexity parameter (here after known as cp).

We will use *train* from *caret* package with cross-validation (k-fold = 20) to illustrate more about the selection of tree sizes. As it is shown on figure 1, for *cp* values of 0.2112 the cross validated accuracy is 0.78 (highest).

Figure 1 shows, the varying cross validated accuracy over different complexity values.

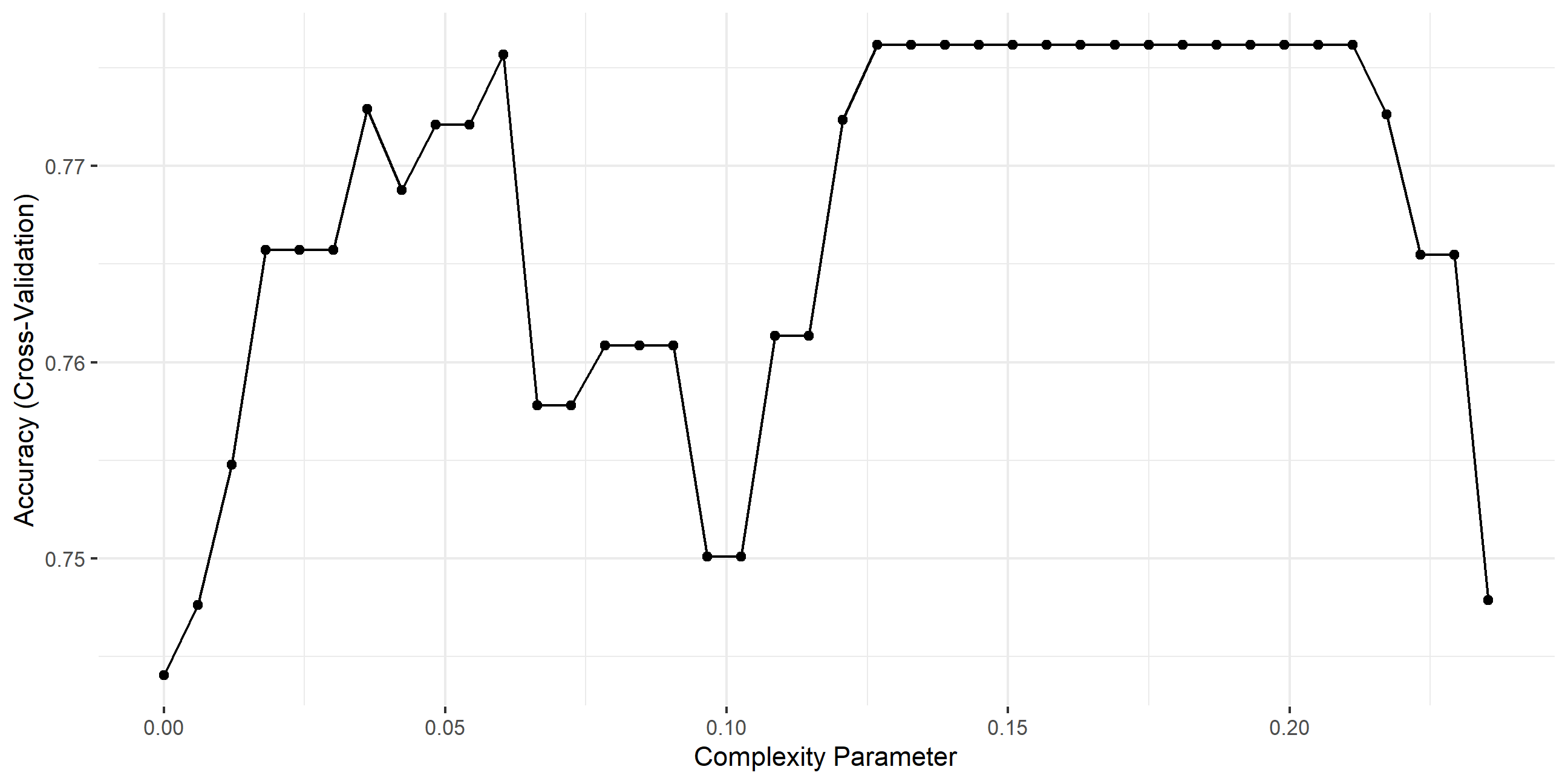


Figure 1: Complexity Vs Misclassification rate

Figure B4 in the supplementary material shows pruned tree with newly gained parameters. Figure B5 in the supplementary also shows the selected predictors.

Table 2: Confusion matrix and accuracy for pruned decision tree

|  | Train Accuracy | Test Accuracy |
| --- | --- | --- |
| Accuracy | 0.81495 | 0.77311 |
| Sensitivity | 0.99531 | 0.95556 |
| Specificity | 0.25000 | 0.20690 |
| Pos Pred Value | 0.80608 | 0.78899 |
| Neg Pred Value | 0.94444 | 0.60000 |
| Prevalence | 0.75801 | 0.75630 |
| Detection Rate | 0.75445 | 0.72269 |

Confusion matrix for train data

| Train | Control | Alizahmer |
| --- | --- | --- |
| Control | 17 | 1 |
| Alizahmer | 51 | 212 |

Confusion matrix for test data

| Test | Control | Alizahmer |
| --- | --- | --- |
| Control | 6 | 4 |
| Alizahmer | 23 | 86 |

As shown in table 2 after pruning the test accuracy is 77%, where as, the train accuracy was 84%. After pruning the variance seemed to decrease. In addition, the test accuracy decreased, this might relate to the small number of remaining predictors after pruning or my tuning strategy for selecting complexity parameter. All in all, the pruned tree is too simple and lost the ability to capture important patterns or relationships in the data.

We will look at several ways to fix this, including bagging, boosting and random forests. Before we do that let’s change gears and look at a linear model.

## Logistic Regression

Linear models (LMs) provide a simple, yet effective, approach to predictive modeling. However, in this high dimentional set, trying to fit simple logistic regression (*glm*), the algorithm will not converge to a solution or find a unique solution. The increase in complexity can result in, where the predictors are highly correlated, leading to ill-conditioned matrices and slow convergence. Nevertheless, since our main goal is to worry less about the dataset, let’s see what happens.

After fitting simple logistic regression, table C1 in the logistic supplementary section shows the fitted coefficients for some predictors (10), and it can clearly be seen for some it is not scientifically meaningful.

In addition, as shown in table C2 of the logistic supplementary section, for the simple logistic regression the test accuracy is 46%, where as, the train accuracy was 100%. As expected, the test accuracy is very small.

## Logistic Regression - regulirized

As previously mentioned, simple logistic regression was not appropriate for solving this problem. Regularization methods offer a way to limit or regulate the estimated coefficients, which can lower variance and decrease error.

In this work I will use generalization of the ridge and lasso penalties, called elastic net. The three parameters we are interested in are: β is the vector of logistic regression coefficients to be estimated; λ is the regularization parameter controlling the strength of regularization; α is the mixing parameter controlling the trade-off between the L1 and L2 penalty terms.

The grid tuning uses cross validation with (*k-fold = 5*) and tune length of 20, and the selected values are generated using *seq(0, 1, by = 0.01)* rule. The are generated using . When performing a grid search, it is often useful to use lambda with a range of values, with increasingly larger gaps between the values as the values become larger. For this task I have used *caret* library. Table 3 shows the selected parameters for the best performing model after tuning.

Table 3: GLM net model specifications cross validation

| alpha | lambda | Accuracy | Kappa | AccuracySD | KappaSD |
| --- | --- | --- | --- | --- | --- |
| 0.01 | 0.27 | 0.94 | 0.83 | 0.023 | 0.073 |

Accordingly, as shown in table 3, the selected parameters are value of 0.01 (close to ridge) and value of 0.27.

Figure C1 in the supplementary section of logistic regression (regularized) shows (plot of mixing percentage () and accuracy) give us a hint in the trade-off between model complexity and accuracy. Typically, a lower value of lambda will result in a more complex model that fits the data well but may over fits. As shown in figure C1 the accuracy is high for low values of lambda but decreases rapidly as lambda increases ( > 0.1).

|  | Train accuracy | Test accuracy |
| --- | --- | --- |
| Accuracy | 0.96441 | 0.97479 |
| Sensitivity | 1.00000 | 1.00000 |
| Specificity | 0.85294 | 0.89655 |
| Pos Pred Value | 0.95516 | 0.96774 |
| Neg Pred Value | 1.00000 | 1.00000 |
| Prevalence | 0.75801 | 0.75630 |
| Detection Rate | 0.75801 | 0.75630 |

Table 4: Confusion matrix and accuracy measure for regularized logistic regression

| Train | Actual Control | Actual Alizahmer |
| --- | --- | --- |
| Control | 58 | 0 |
| Alizahmer | 10 | 213 |

| Test | Actual Control | Actual Alizahmer |
| --- | --- | --- |
| Control | 26 | 0 |
| Alizahmer | 3 | 90 |

As shown in table 4, the test accuracy for this model is 97%, where as, the train accuracy was estimated to be 96%. Compared to the simple logistic regression model this regularized elastic model does much better in-terms of accuracy and reducing the risk of over-fitting. For such a simple model this is an incredible accuracy level.

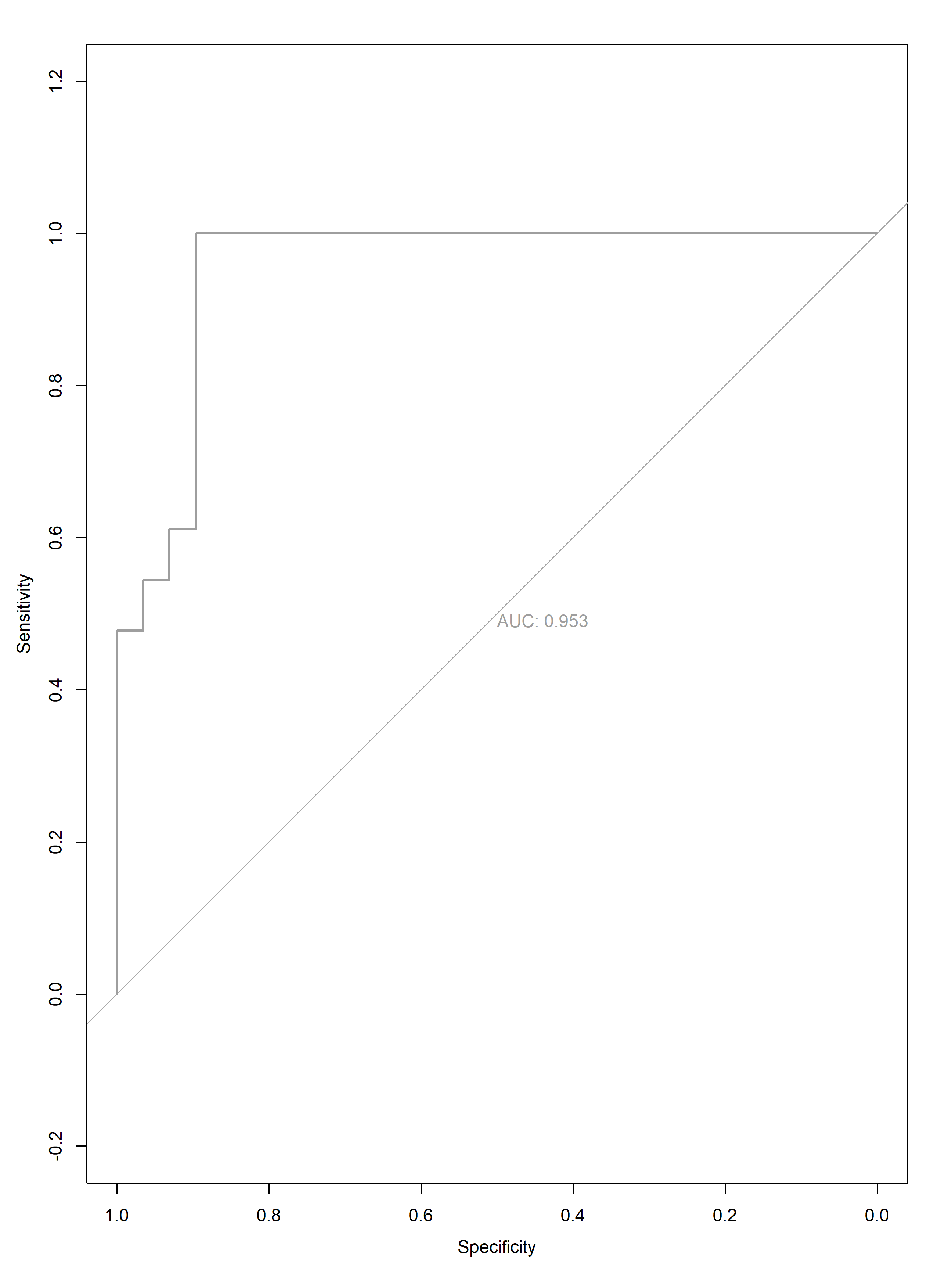


Figure 2: ROC curve for GLM net fit (AUC: Area under the curve – 95.3%)

As shown in figure 2, the ROC for elastic model with AUC of 95.29%. Figure C2 in supplementary material shows the selected feature for this model. This also shows logistic regression will work best with feature engineering tools like Principal Component Analysis (PCA).

Check additional description for this model in the supplementary section.

## kNN

kNN is framed as a non-parametric model for the probabilities . It models *k* neighbors estimates this probability as

Essentially, the probability of each class *g* is the proportion of the *k* neighbors of x with that class, g. Then to create a classifier we use:

In this work since we are focused on binary outcome, the classifier can be re-written as:

To avoid large numbers impacting other variables distance measurement, we will scale the predictors (mean zero and standard deviation of 1). For selecting which *k* value to use, we will try many options and select the one with lowest error rate and smaller chance of overfitting.

For k values (from 1 to 20) kNN model was trained. Table D1 in the supplementary, shows the test and train accuracy for 20 different *k* values. Generally, *k* values from 10 to 17 can be used as the difference between test and train accuracy is small and they have the highest test accuracy.

Figure 3 also shows the tradeoff plot between test and train accuracy with varying k values. As it is shown in the figure xx the test and train accuracy will start decreasing after *k = 19* (we tested 100 k-values, from 1 to 100). *k = 10, 11, 12, 13, 14* has the highest test accuracy, which is also close to train accuracy. However, *k = 10* or *k = 12* has the highest test accuracy, but it tends to underfit.

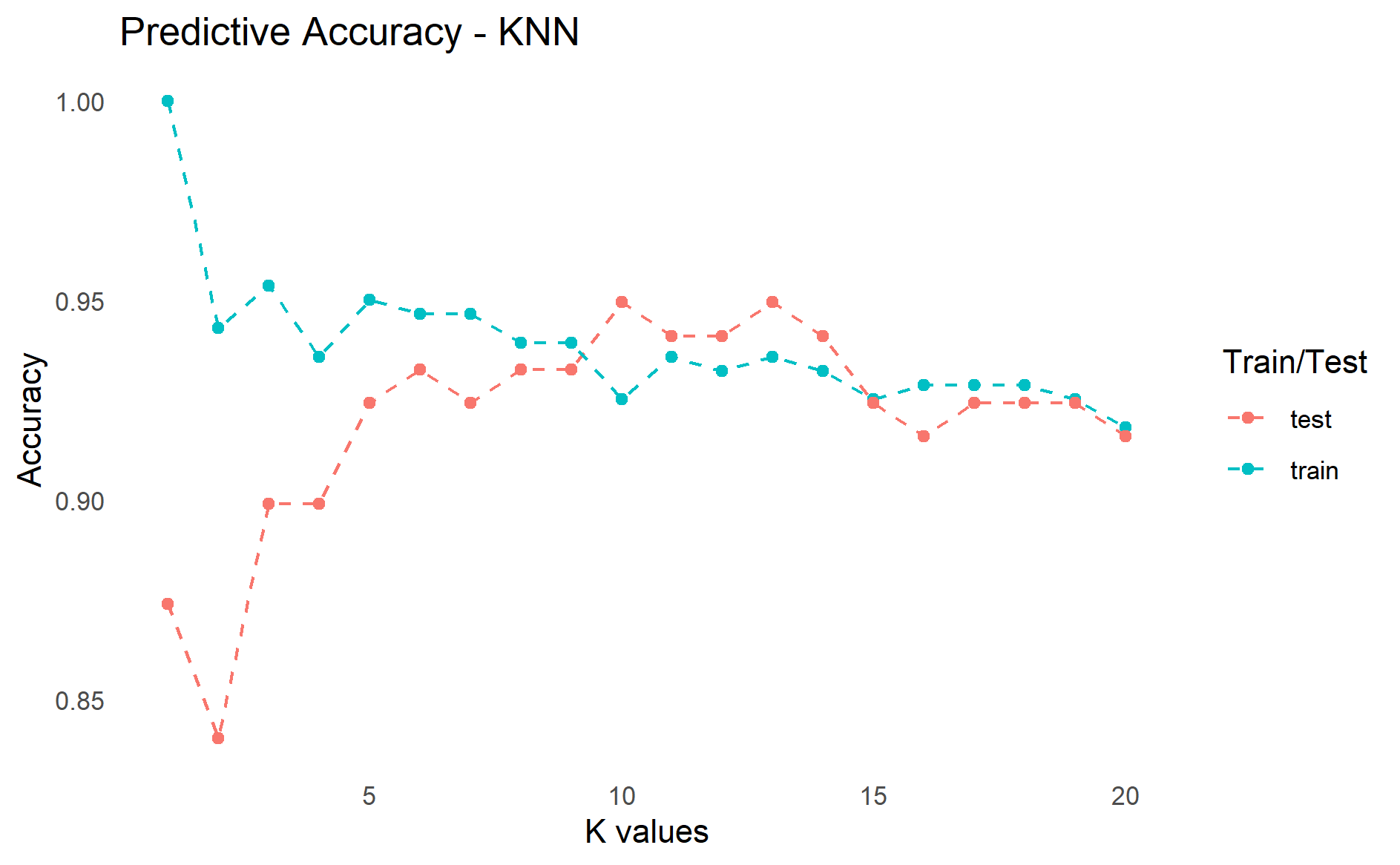


Figure 3: Train-test accuracy

Figure D1 in the supplementary section for kNN also shows test error rates with varying *k* values. The dotted orange line represents the smallest observed test classification error rate. We can fit our *knn* model using *k = 12*.

Finally, as shown in table D1 of the supplementary material for kNN, the train accuracy related to *k* value of 12 is 93% and the test accuracy is 94%.

In the above model we used *knn* model from *class* library. We can also use cross validation for selecting *k* values using *caret* library. In implementing with caret, center and scale was used for scaling. The center subtracts the mean of the predictor’s data from the predictor values while scale divides by the standard deviation. Accordingly, this transforms the data into mean of zero and standard error of one. *k-fold* of 10 with 3 repitation and tune length of 20 were used.

Table 5: Confusion matrix and accuracy measure for kNN (after cross validation)

| Train | Actual Control | Actual Alizahmer |
| --- | --- | --- |
| Control | 54 | 0 |
| Alizahmer | 14 | 213 |

|  | Train accuracy | Test accuracy |
| --- | --- | --- |
| Accuracy | 0.95018 | 0.94958 |
| Sensitivity | 1.00000 | 0.98889 |
| Specificity | 0.79412 | 0.82759 |
| Pos Pred Value | 0.93833 | 0.94681 |
| Neg Pred Value | 1.00000 | 0.96000 |
| Prevalence | 0.75801 | 0.75630 |
| Detection Rate | 0.75801 | 0.74790 |

| Test | Actual Control | Actual Alizahmer |
| --- | --- | --- |
| Control | 24 | 1 |
| Alizahmer | 5 | 89 |

Figure D2 in the supplementary material shows, varying cross validated accuracy with varying k values. Table D2 in the supplementary material shows the selected parameters, which resulted in smallest error. Accordingly, using the selected value of k = 5 after cross validation then training and predicting kNN model results in values shown in Table 5.

Chart

Description automatically generated

Figure 4: ROC curve for kNN (AUC: Area under the curve – 93.5%)

# Ensemble models - Bagging

## Random forest

Random forests are an adapted version of bagged decision trees that construct a vast set of uncorrelated trees to enhance predictive accuracy. This learning algorithm has gained widespread popularity due to its good predictive performance with minimal hyperparameter tuning. During the bagging process, while building a decision tree, random forests employ split-variable randomization. This method restricts the search for the split variable to a random subset of of the original *p* values whenever a split is required. Typical value of for classification is . For the first part, I will use *randomForest* library, which we have seen in class.

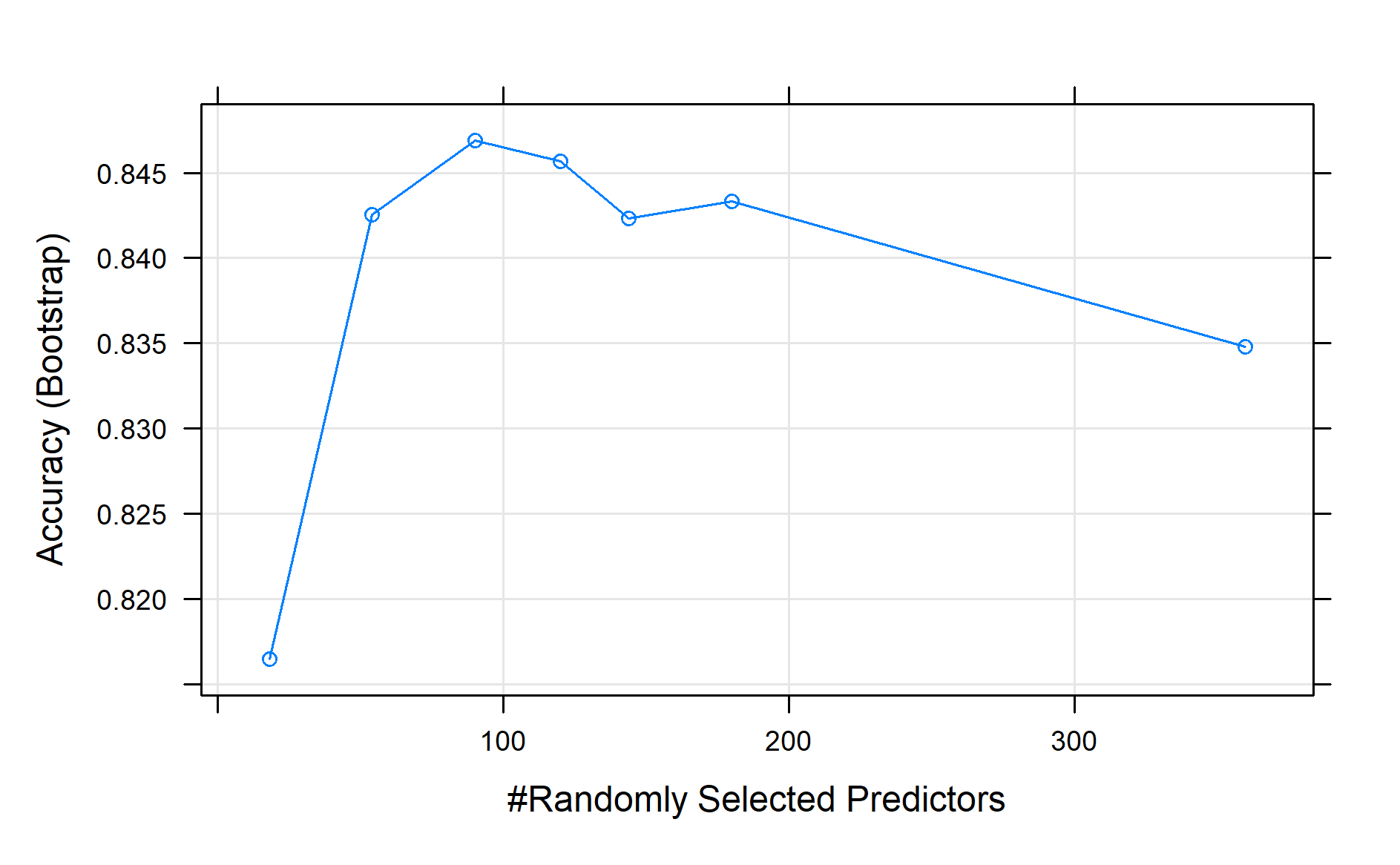
Using this value of the test accuracy was calculated to be 83.19 and the train accuracy was 100. There is a significant sign of over fitting. In addition, the under performance of random forest as compared to simpler methods that we have seen above talks as about the linearity of the given data.

Let’s try tuning. We will be using *caret* library for this. The hyper parameters, which we will be focusing on is:

* **The number of features to consider at any given split:**  This parameter tuning has the highest impact on the model’s accuracy. In this project we will use a 5, 15, 33, 40, 50, 90, and 100% of our number of predictors (this will take 45 minutes to run). A grid search on all potential values of was used, i.e., from 1 to 360, but on my personal laptop it takes more than 9 hours (I stopped it at the ninth hour).

We can also tune number of trees (not a hyperparameter per se), the complexity of each tree, sampling scheme, and splitting rule, however, these have only marginal effect in increasing the accuracy of the model.

Figure 5 shows different bootstrapped accuracy over varying randomly selected predictors. As shown in the figure of 90 was selected and the corresponding accuracy is 84.7%.

Figure 5: Accuracy (bootstrap) versus mtry for random forest

As shown in the table E1 of the supplementary section for random forest the test accuracy of this tuned model is 86%. The accuracy increased as compared to the un-tuned model. The train accuracy is 100%. Figure E1 in the supplementary section of random forest shows the top ten selected features.

# Boosting

There are various supervised machine learning algorithms that use a single predictive model, such as ordinary linear regression, penalized regression models, and single decision trees. In contrast, bagging and random forests combine multiple models to create an ensemble, which improves the accuracy of predictions. Bagging (and random forests) work best with models that have high variance and low bias, like an overgrown decision tree. Boosting is a general algorithm for creating an ensemble using simpler models, usually decision trees. Boosting is most effective with models that have low bias and high variance.

## Simple GBM

Gradient Boosting Machines (GBMs) are a highly renowned and widely adopted machine learning algorithm that has demonstrated remarkable efficacy across a variety of application domains. In contrast to Random Forest, GBMs create an ensemble of weak or shallow trees in a sequential manner, whereby each subsequent tree improves on the predictive power of the prior tree. For the simple GMB we will use *gbm* function (distribution = Bernoulli), with n.trees = 2000, interaction.depth = 6, and shrinkage = 0.01. With CV k-fold of 10, this took 3 minutes.

For this boosting model the train accuracy is 100%, whereas the test accuracy is 92%. Well improved performance as compared to random forest.

Figure F1 in the supplementary section for simple GBM shows a cross validated error and n trees. n tree of 746 seems an optimum point.

We can also try extensive grid search for tuning some of boosting hyperparameters (using *caret*). I used the following for tuning the parameters: interaction depth = 1 to 7; number of trees from 500 to 3,000; shrinkage = 0.5, 0.3, 0.1, 0.05, 0.01; and the minimum number of observations in a node of the tree 5, 10, 15, 20. The algorithm took **12 hours** to complete. Rather than tuning all parameters together, for randomly selected value of learning rate (usually high value) we can find best number of trees. Then we can tune the learning rate by using the newly acquired number of tree while keeping others constant. After getting the learning rate we can tune other tree specific parameters (a lot of work, but faster).

Chart

Description automatically generated

Figure 6: Plot for cross validated accuracy for set hyperparameters. Each line represents a boosting iteration of 500, 1000, 1500, 2000, 2500 and 3000. Each box represents different combination of minimum number of observations in a node and shrinkage value.

In this work all the parameters are defined together and will be tuned together. Overall, for these defined values 642 models were tried.

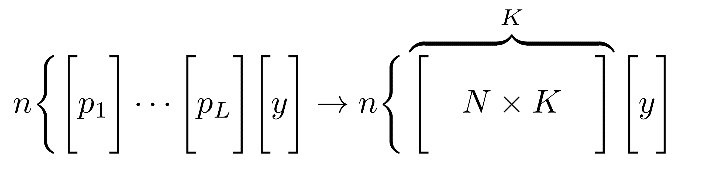
Table F1 in the supplementary section for simple GBM shows the selected parameters for after tuning.

As it is shown in table F2 of the supplementary section for simple GBM (tuned), the train accuracy is 100%, where as, the test accuracy is 93%. Slight increment as compared to the un-tuned GBM in-terms of accuracy.

# Stacking of existing models

Stacking, is designed to ensemble a diverse group of strong learners. The first potential candidates were all the models we have seen above. However, I have removed random forest as it reduced the overall accuracy of the stacked model. For each candidate base learners, we can go through hyperparameter tuning while building the stacked model. Fearing this might take a very long time (days, maybe) I decided to use default tuning setup provided in *caret* in building my ensemble. Parallel programming is an option, but this quarter drained me of the motivation and energy to do so. For stacking we will use *caretEnsemble and caretList* library. The steps are as follow:

* Train the base models (regularized logistic regression - *glmnet*, simple boosting - *gbm*, and k Nearest Neighbors - *knn*) with tuned parameters (but the tuning was across default value).
* Perform k-fold CV on each of the base learners and collect the cross-validated predictions (N) from each (the same k-folds must be used for each base learner).
* Create a data using K times cross-validated predicted values (N) along with our response (y), where K is the number of base learner models.



Where n is observation number

* Train the meta model, in this case we will use simple model which is regularized logistic regression implemented by *glmnet* ) on the above data matrix. For training the meta model we will use data, y = f(N x K).

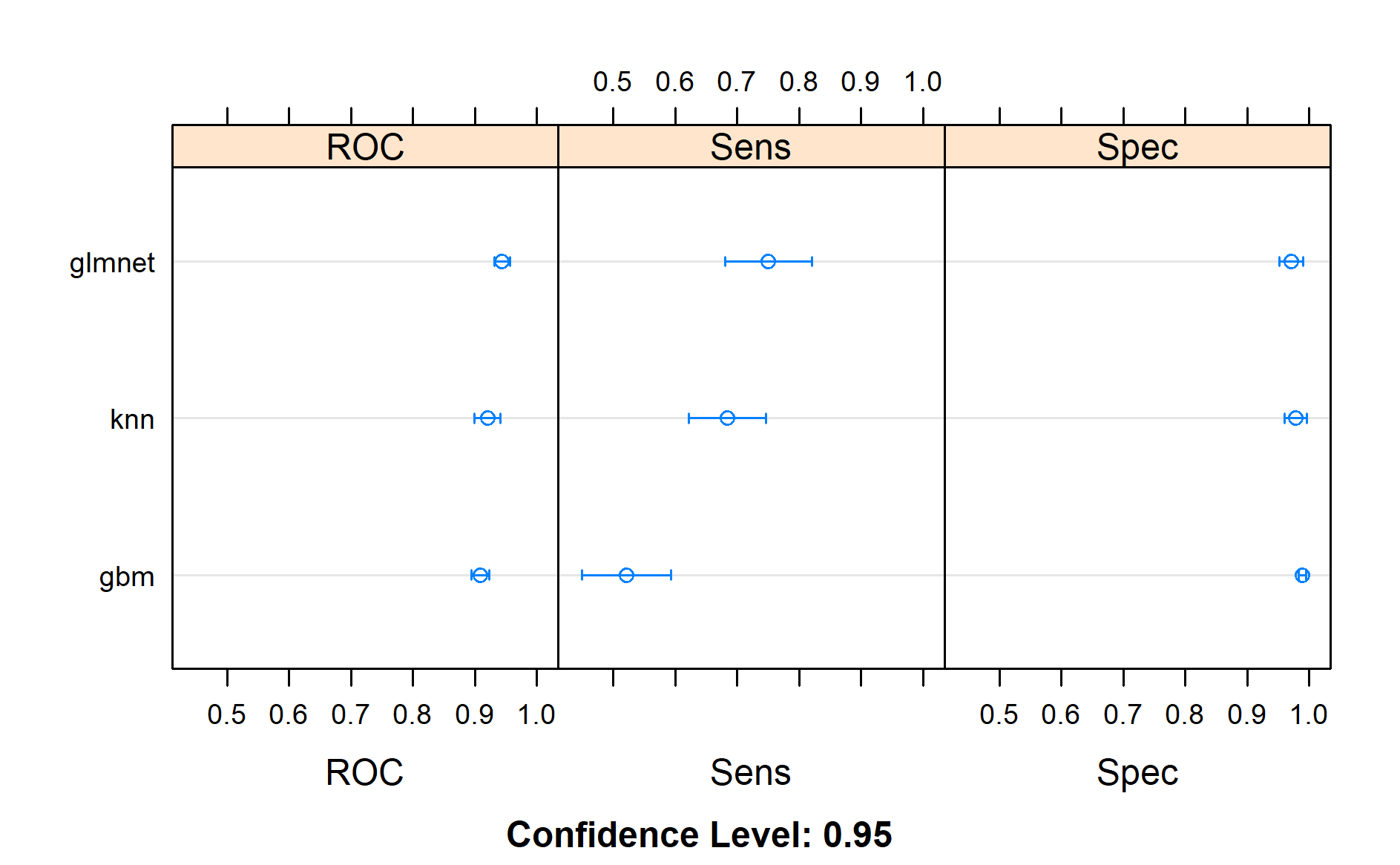


Figure 7: ROC, specificity, and sensitivity plot for the base learners

Figure 7 shows the dot plot showing ROC, sensitivity and specificity of the base learners model. As it is clearly shown the simple model (elastic net), i.e, *glmnet* outperformed the much complex model as boosting. The correlation plot and matrix of these three models is shown in table G1 and figure G1 of the supplementary material section of stacking. The accuracy of the stacked model as shown in table G2 of the supplementary section is 96%. Since the base learners are correlated (*glmnet and kNN)* no accuracy gain was observed. However, as compared to *glmnet* the specificity increased. This is a significant gain as the control numbers are very small. Figure G2, of the supplementary section for simple GBM also shows the first 10 important features for this model.

# Conclusion

Let’s put everything together:

Table 6. Summary of all the models

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Test Accuracy | AUC | Sensitivity | Specificity | Remark |
| Elastic net (glmnet) | 97% | 95% | 100% | 90% | Underfits |
| kNN | 95% | 93% | 99% | 83% | Overfits and decreased specificity |
| Tuned random forest | 86% | - | 99% | 45% | Overfits, low test accuracy, low specificity and low AUC |
| Tuned boosting | 92% | 88% | 94% | 83% | Overfits, low AUC and specificity |
| Stacked model | 96% | 95% | 97% | 93% | Doesn’t overfit, increased specificity, high sensitivity, and high AUC. |

As it is shown in the above table the simple model (elastic net) has the highest accuracy. Much of the strength of the stacking models also comes from this model. All in all, with increased specificity and the accuracy of the stacked model being the same as elastic net the stacked model is seems to be the best model for the different scenarios we discussed in this report. Looking again to figure A3 in the appendix, we can clearly see this data can be defined or described by linear models.

# Supplementary materials

## Section A: Data

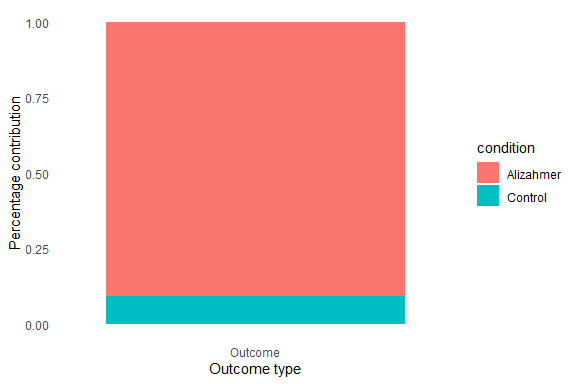
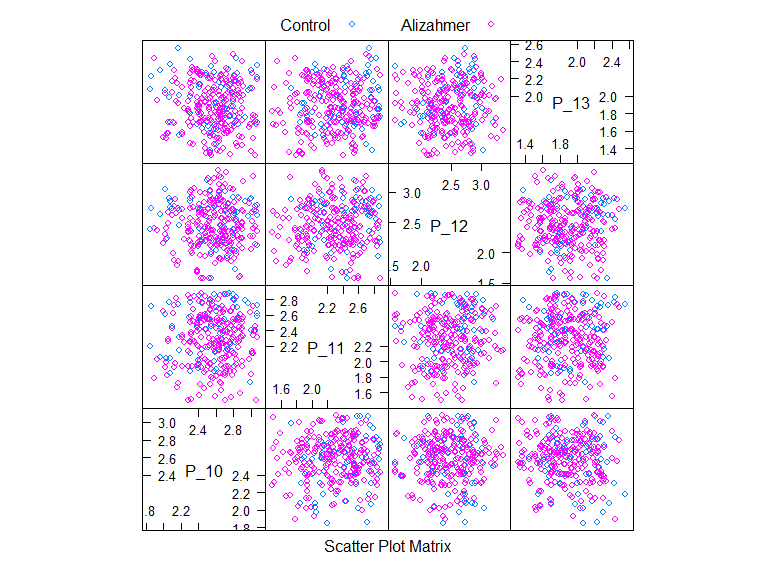


Figure A1: Outcome groups

Chart, scatter chart

Description automatically generated

Figure A2: Scatter plot of the data for three predictors (P\_10, P\_11, P\_12, and P\_13)

Figure A3: PCA reduced to two variables and plotted

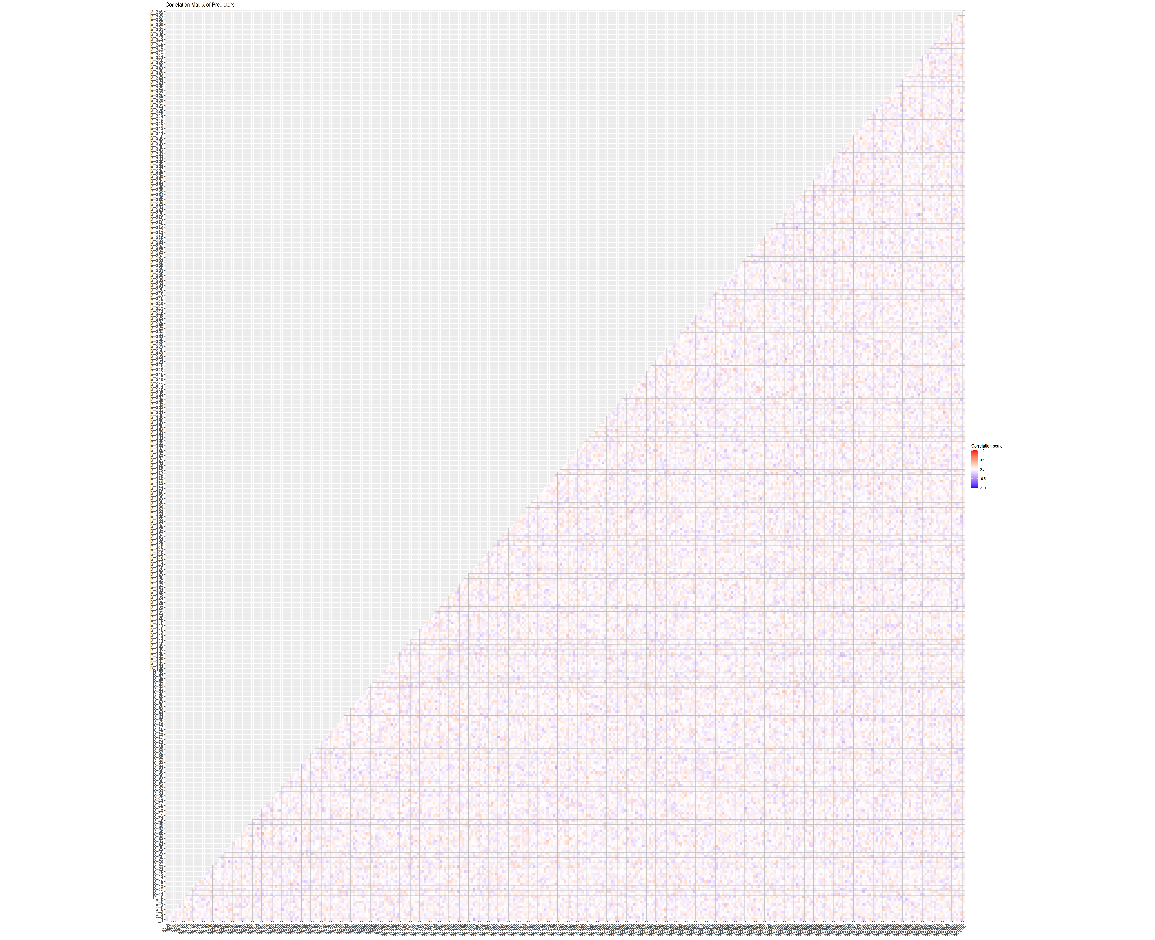


Figure A4: Correlation matrix for our train data

## Section B: Simple tree

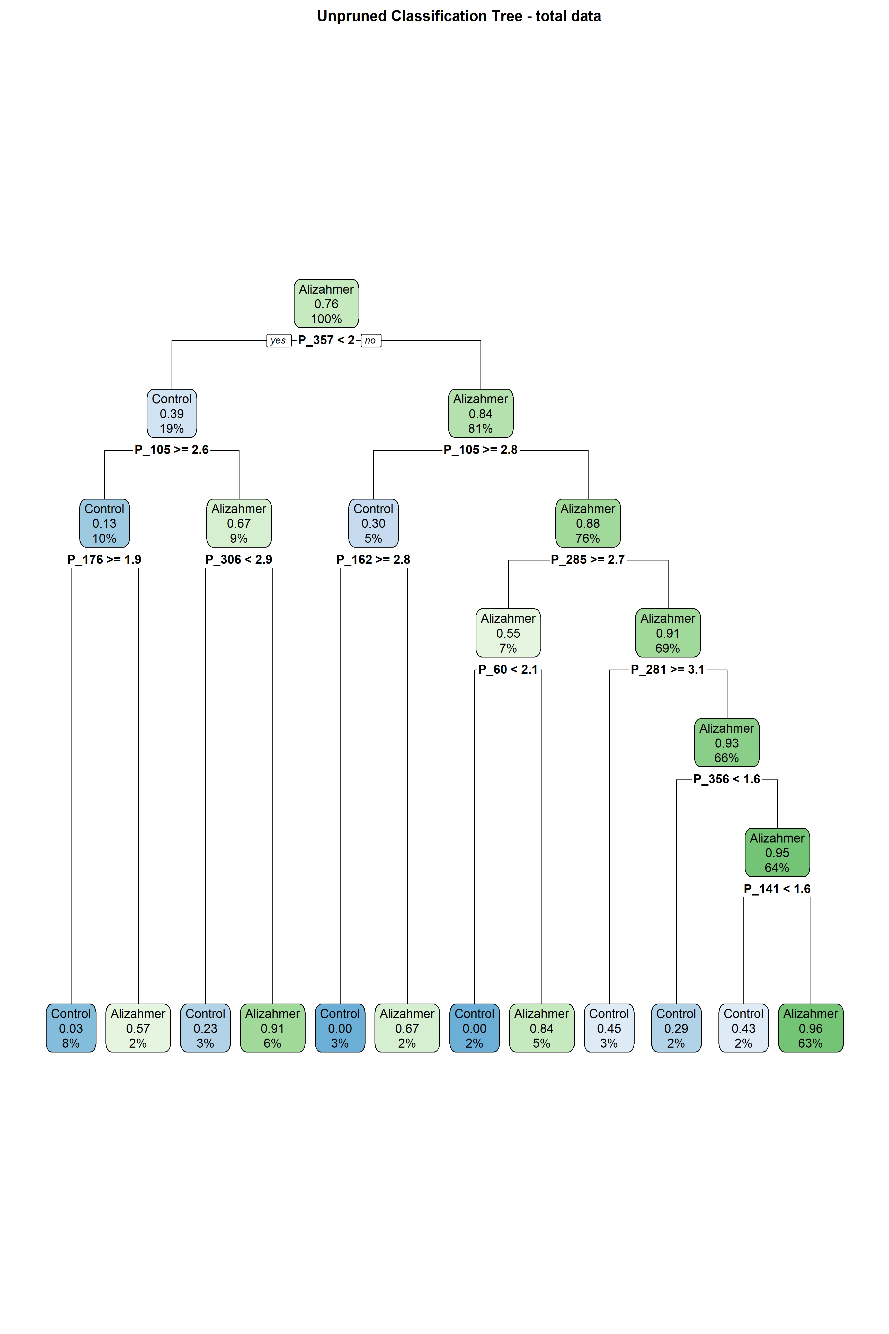


Figure B1: Full grown tree of the data

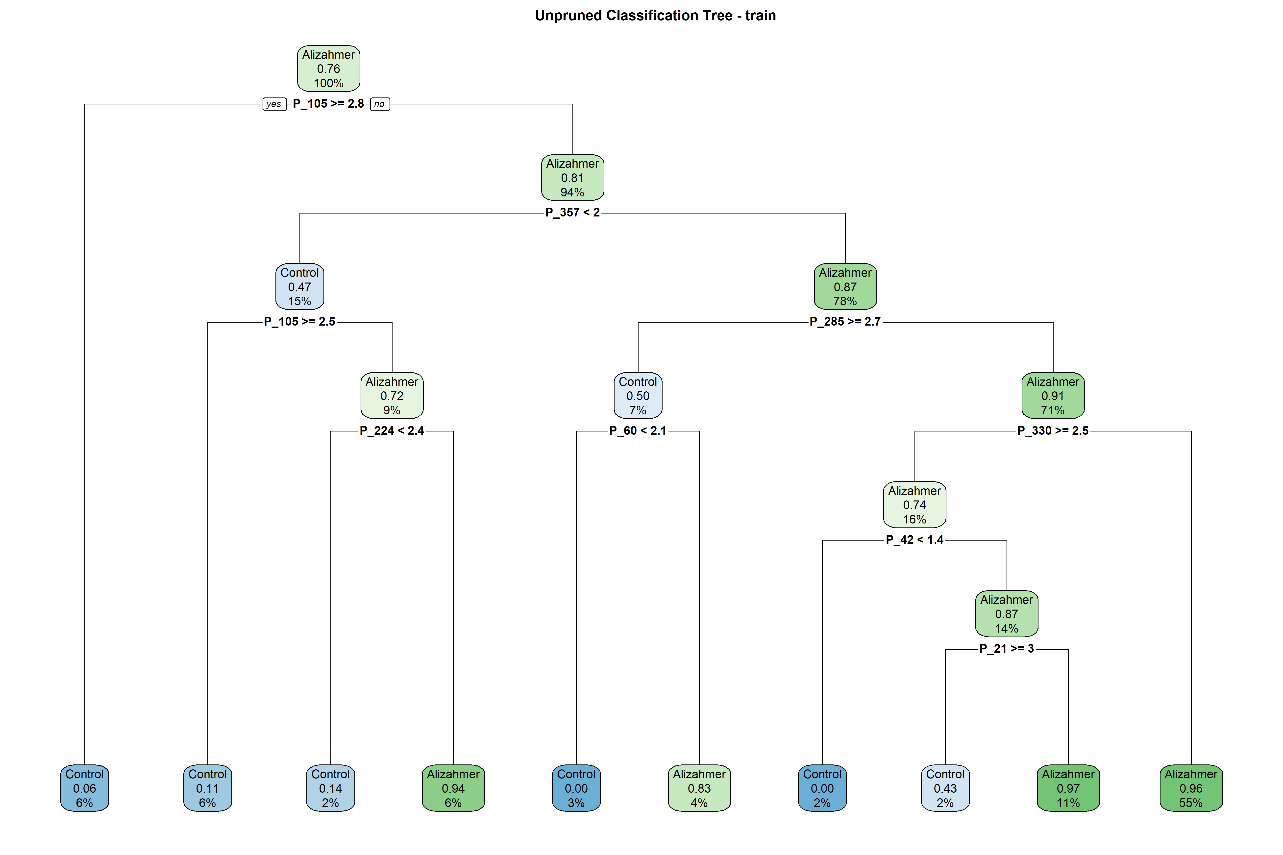
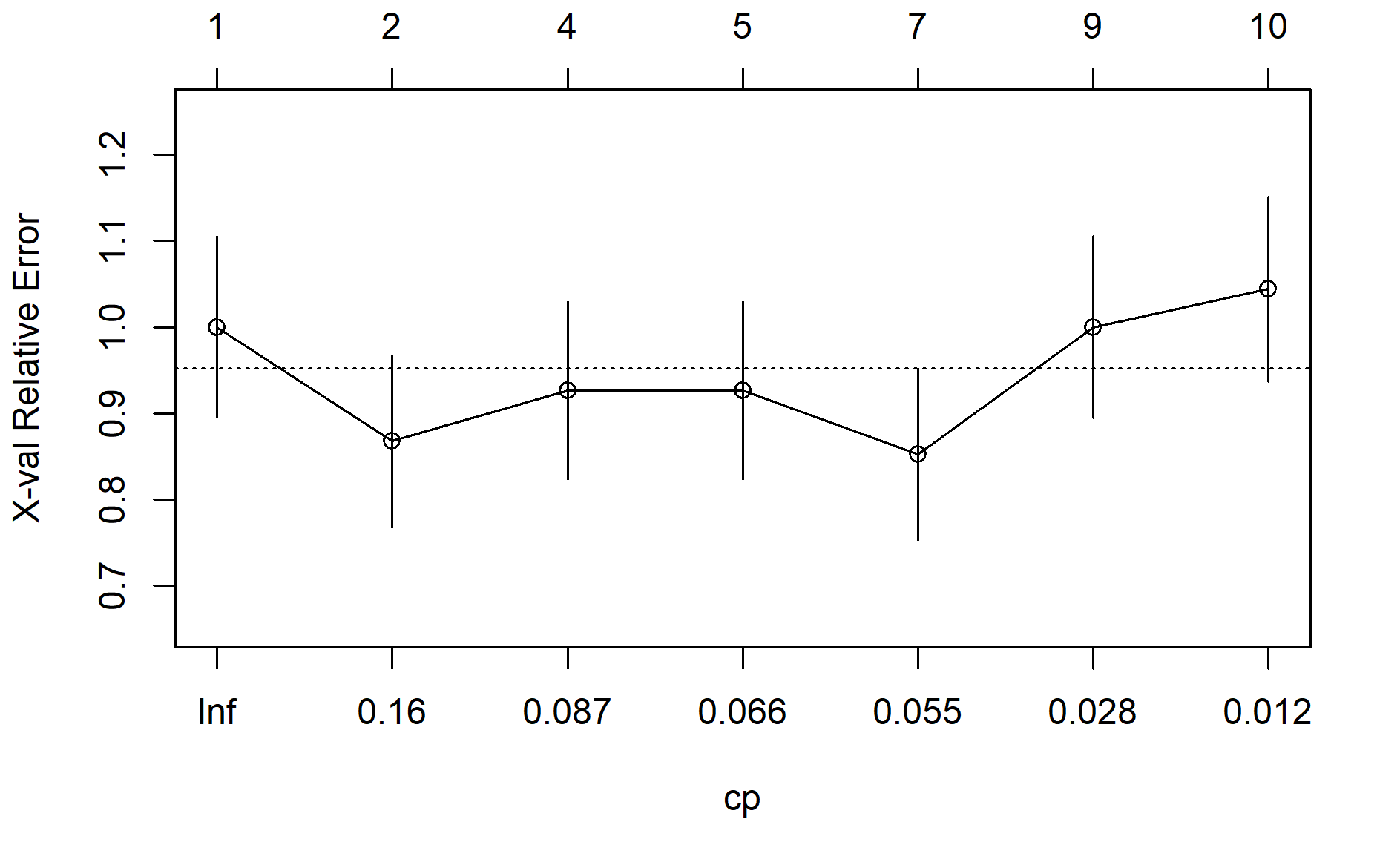


Figure B2: Full grown tree of train data

Diagram

Description automatically generated

Figure B3: Cross validated accuracy versus changing complexity parameters

Figure B4: Pruned tree using cross validation

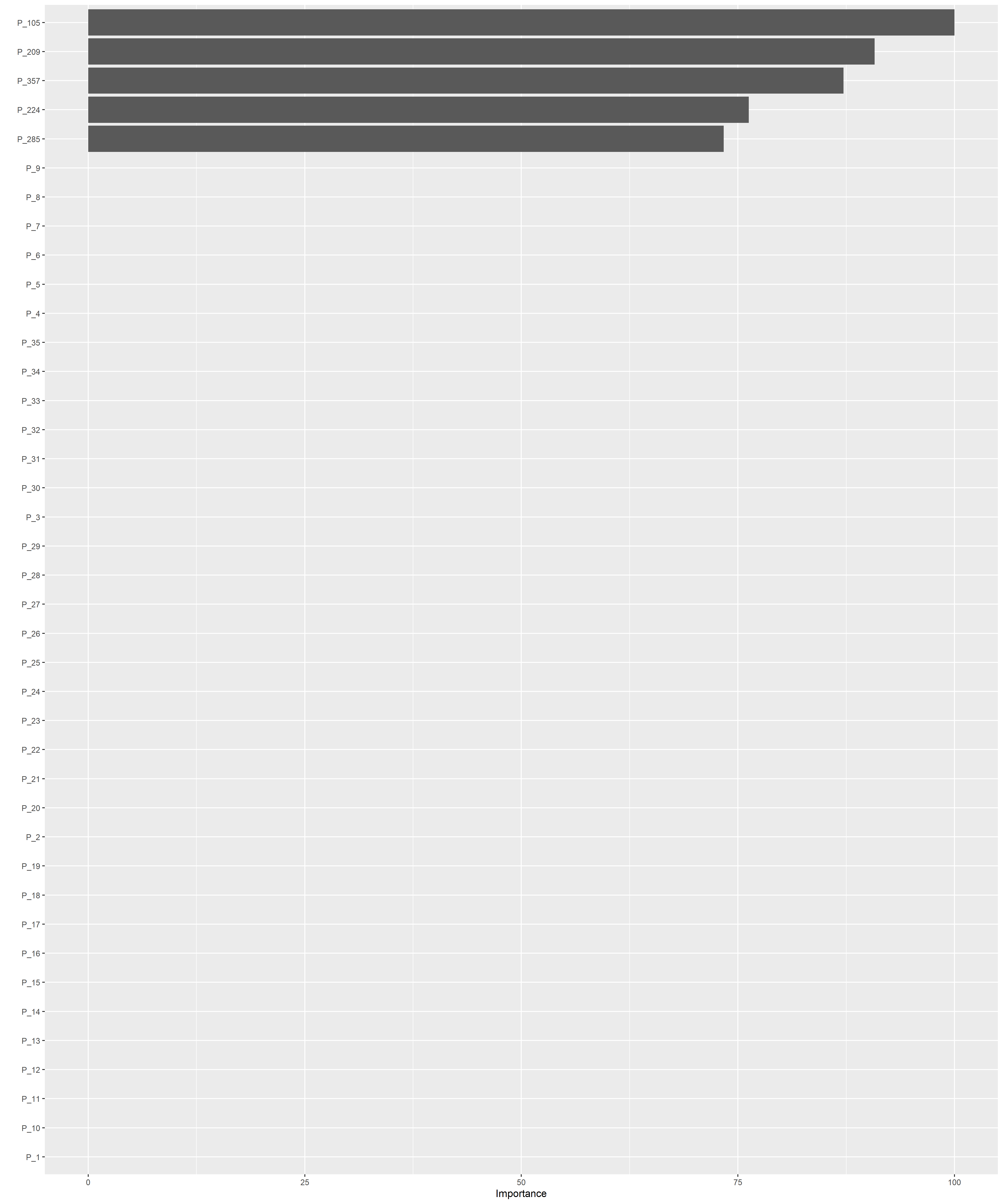


Figure B5: Features selected after pruning (5 predictors left)

## Section C: Logistic Regression

Table C1: Logistic regression training coefficients (for the first 10 predictors)

|  | Estimate | Std. Error | z value | Pr(>|z|) |
| --- | --- | --- | --- | --- |
| (Intercept) | 20320.04 | 455838924 | 0 | 1 |
| P\_1 | -1637.76 | 30297377 | 0 | 1 |
| P\_2 | -1800.36 | 36603425 | 0 | 1 |
| P\_3 | -1120.94 | 21664864 | 0 | 1 |
| P\_4 | -122.19 | 2179335 | 0 | 1 |
| P\_5 | -840.98 | 17657953 | 0 | 1 |
| P\_6 | 19.54 | 1657746 | 0 | 1 |
| P\_7 | 767.83 | 13290991 | 0 | 1 |
| P\_8 | 838.44 | 17133000 | 0 | 1 |
| P\_9 | 687.15 | 12108407 | 0 | 1 |

Table C2: Confusion matric and accuracy values for logistic regression

| Train | Actual Benign | Actual Malignant |
| --- | --- | --- |
| Control | 68 | 0 |
| Alizahmer | 0 | 213 |

|  | Train accuracy LR | Test accuracy LR |
| --- | --- | --- |
| Accuracy | 1.00000 | 0.46218 |
| Sensitivity | 1.00000 | 0.48889 |
| Specificity | 1.00000 | 0.37931 |
| Pos Pred Value | 1.00000 | 0.70968 |
| Neg Pred Value | 1.00000 | 0.19298 |
| Prevalence | 0.75801 | 0.75630 |
| Detection Rate | 0.75801 | 0.36975 |

| Test | Actual Benign | Actual Malignant |
| --- | --- | --- |
| Control | 11 | 46 |
| Alizahmer | 18 | 44 |

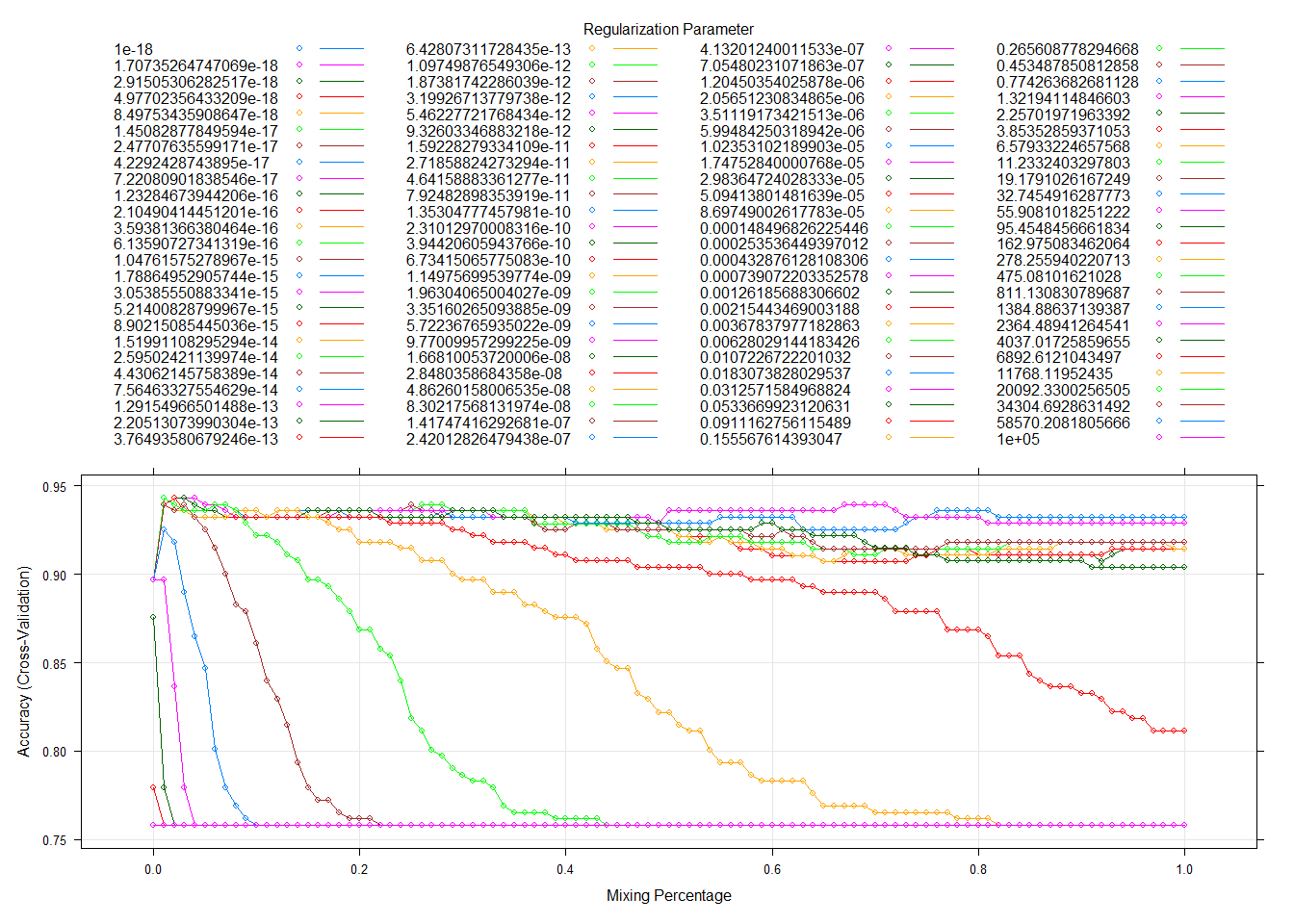


Figure C1: GLM net cross validation

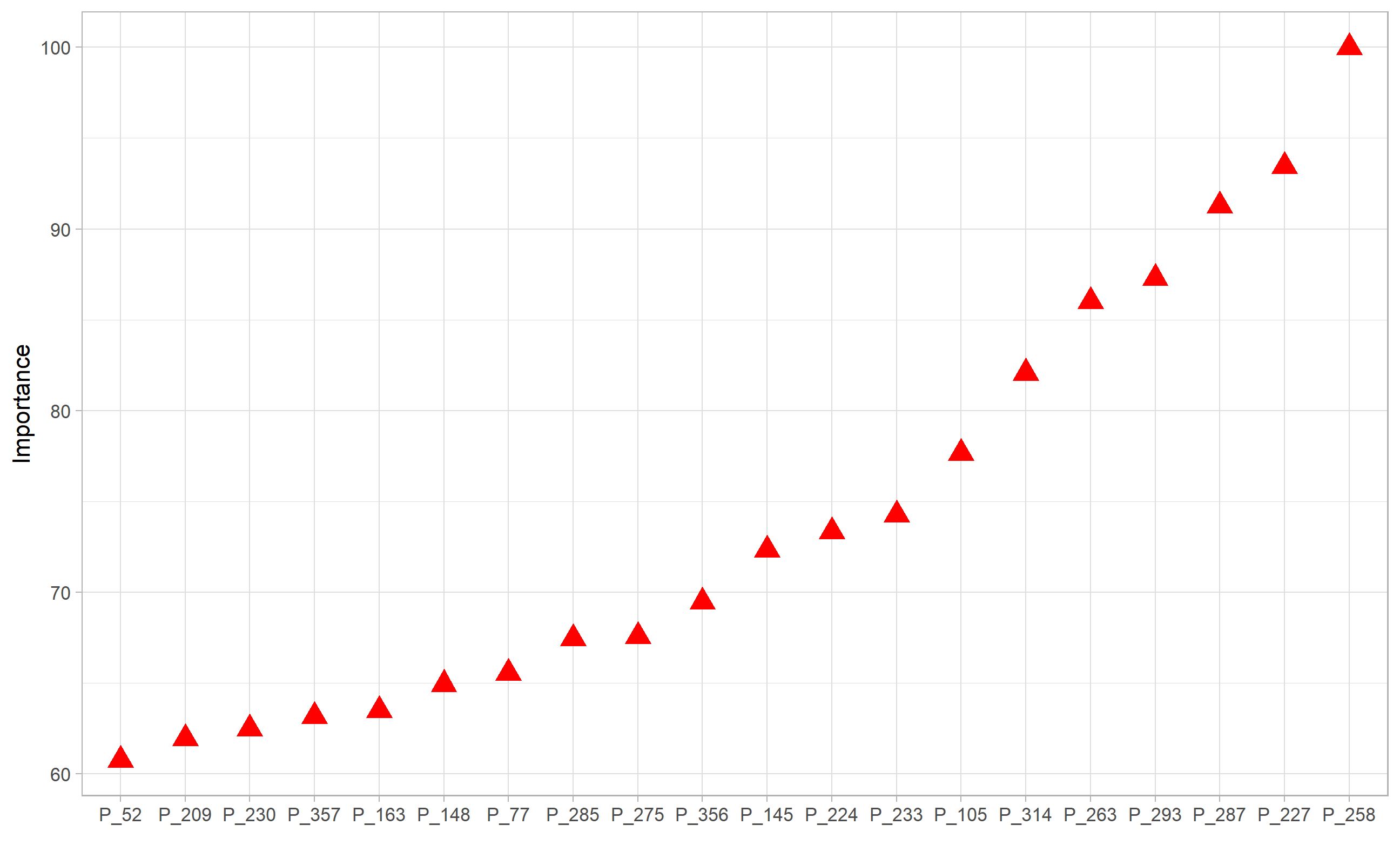


Figure C2: Features extracted from through GLM net - elastic

***Additional modelling information***

For providing additional information and analysis on the use of elastic net, I will use the *glmnet* function as it is more flexible.

I fitted glmnet (family of *binomial*) using the given different values. As we have seen above the value of the elastic net is close to ridge, so we will use of 0 in this exploration.

Figure C3 shows all coefficient values for the selected and values.

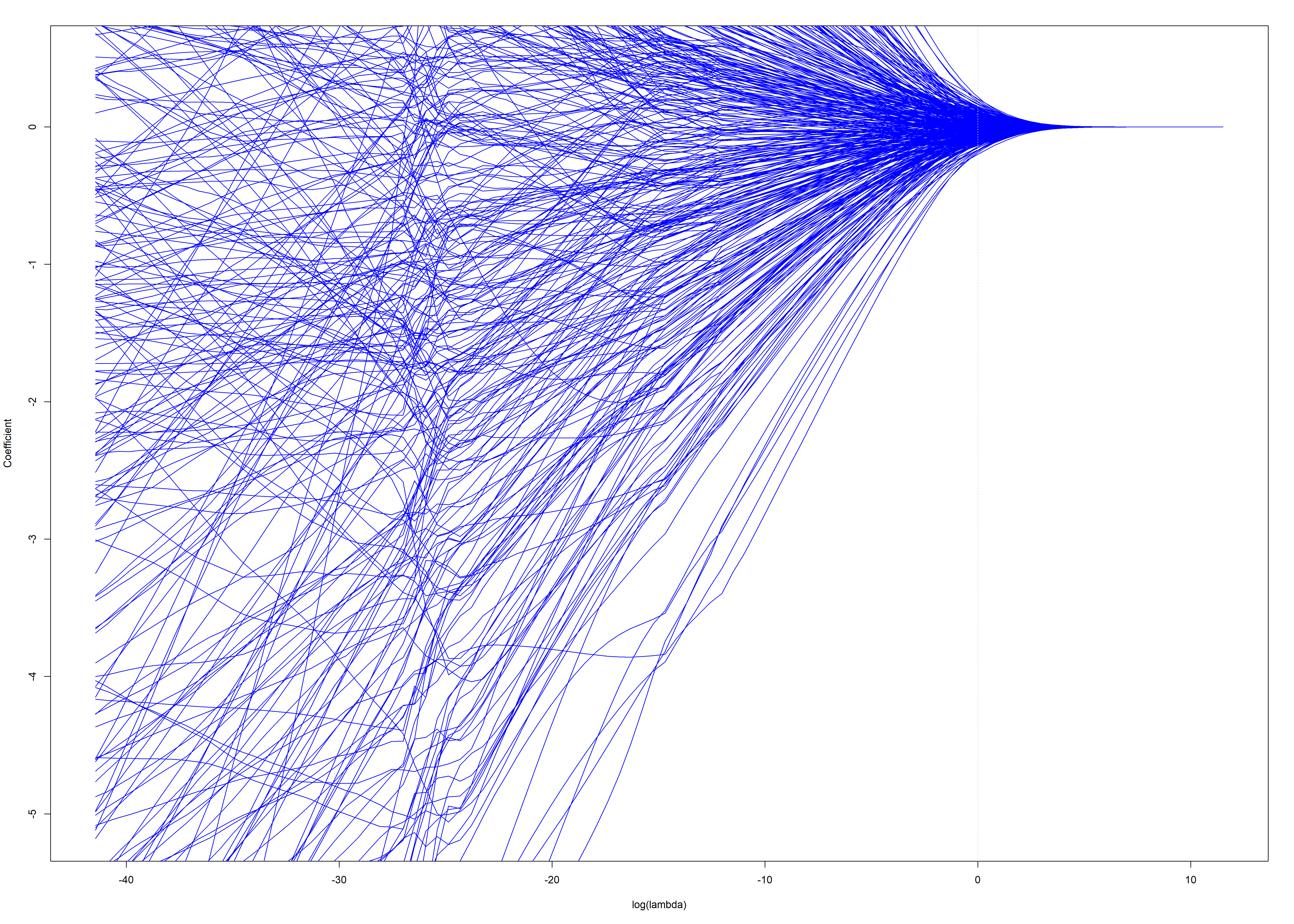


Figure C3: log(lambda) Vs beta values for all the predictors - simple ridge fit (looks weird)

Figure C4 also shows the mis-classifaction error and . The mis-classification error is calculated based on the cross-validation results, and controls the strength of regularization applied to the model. The goal is to choose a value of that balances regularization with model performance on new data. This figure shows how mis-classifications error changes with increasing value and where this potential value is.

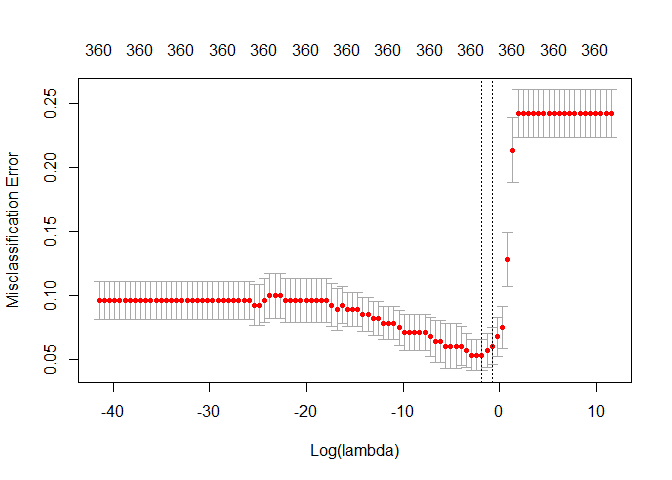


Figure C4: Misclassification Error vs. Log(lambda)

## Section D: kNN

Table D1: Predictive accuracy - kNN

| k | training | test |
| --- | --- | --- |
| 1 | 1.000 | 0.874 |
| 2 | 0.943 | 0.840 |
| 3 | 0.954 | 0.899 |
| 4 | 0.936 | 0.899 |
| 5 | 0.950 | 0.924 |
| 6 | 0.947 | 0.933 |
| 7 | 0.947 | 0.924 |
| 8 | 0.940 | 0.933 |
| 9 | 0.940 | 0.933 |
| 10 | 0.925 | 0.950 |
| 11 | 0.936 | 0.941 |
| 12 | 0.932 | 0.941 |
| 13 | 0.936 | 0.950 |
| 14 | 0.932 | 0.941 |
| 15 | 0.925 | 0.924 |
| 16 | 0.929 | 0.916 |
| 17 | 0.929 | 0.924 |
| 18 | 0.929 | 0.924 |
| 19 | 0.925 | 0.924 |
| 20 | 0.918 | 0.916 |

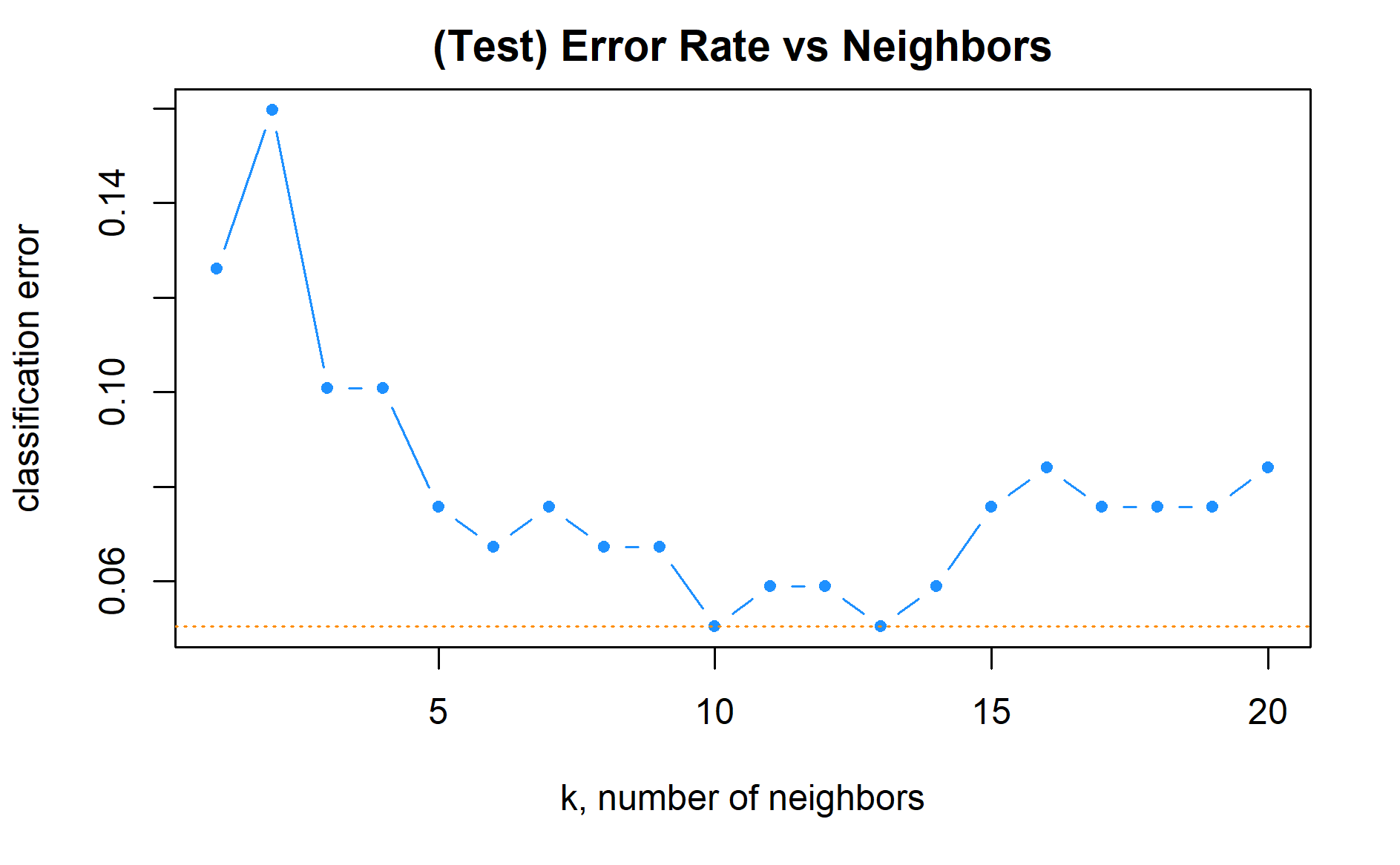


Figure D1: Cross validated *k* values vs cross-validation error.

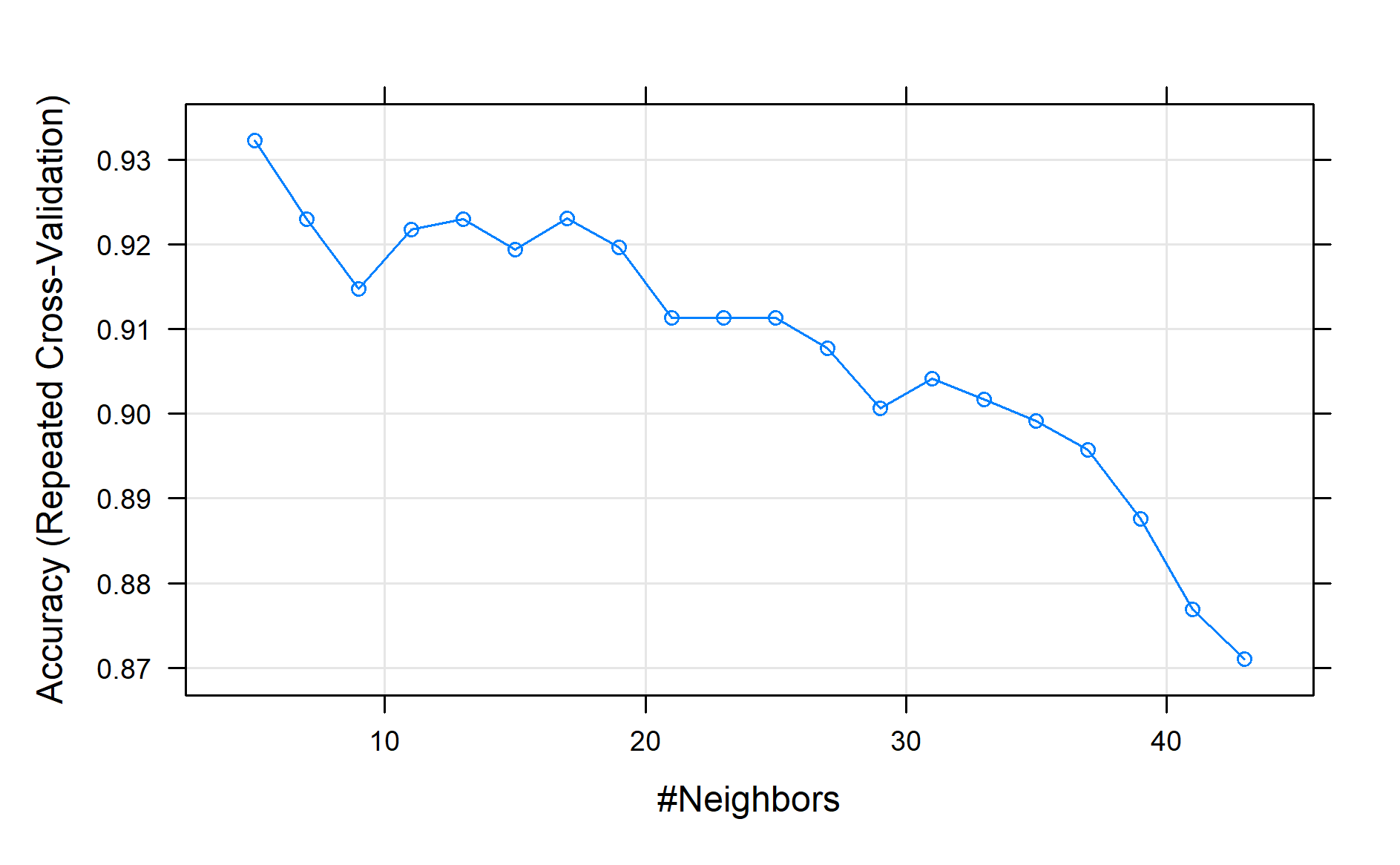


Figure D2: kNN model specifications cross validation

Table D2: Selected parameters after cross validation using caret (kNN)

| k | Accuracy | Kappa | AccuracySD | KappaSD |
| --- | --- | --- | --- | --- |
| 5 | 0.9347169 | 0.8006882 | 0.0429419 | 0.1344253 |

## Section E: Random Forest

| Train data | Actual Control | Actual Alizahmer |
| --- | --- | --- |
| Control | 68 | 0 |
| Alizahmer | 0 | 213 |

Table E1: Confusion matrix and accuracy values for Random Forest model

|  | Train accuracy |  | Test accuracy |
| --- | --- | --- | --- |
| Accuracy | 1.00000 |  | 0.85714 |
| Sensitivity | 1.00000 |  | 0.98889 |
| Specificity | 1.00000 |  | 0.44828 |
| Pos Pred Value | 1.00000 |  | 0.84762 |
| Neg Pred Value | 1.00000 |  | 0.92857 |
| Prevalence | 0.75801 |  | 0.75630 |
| Detection Rate | 0.75801 |  | 0.74790 |

| Test data | Actual Control | Actual Alizahmer |
| --- | --- | --- |
| Control | 13 | 1 |
| Alizahmer | 16 | 89 |

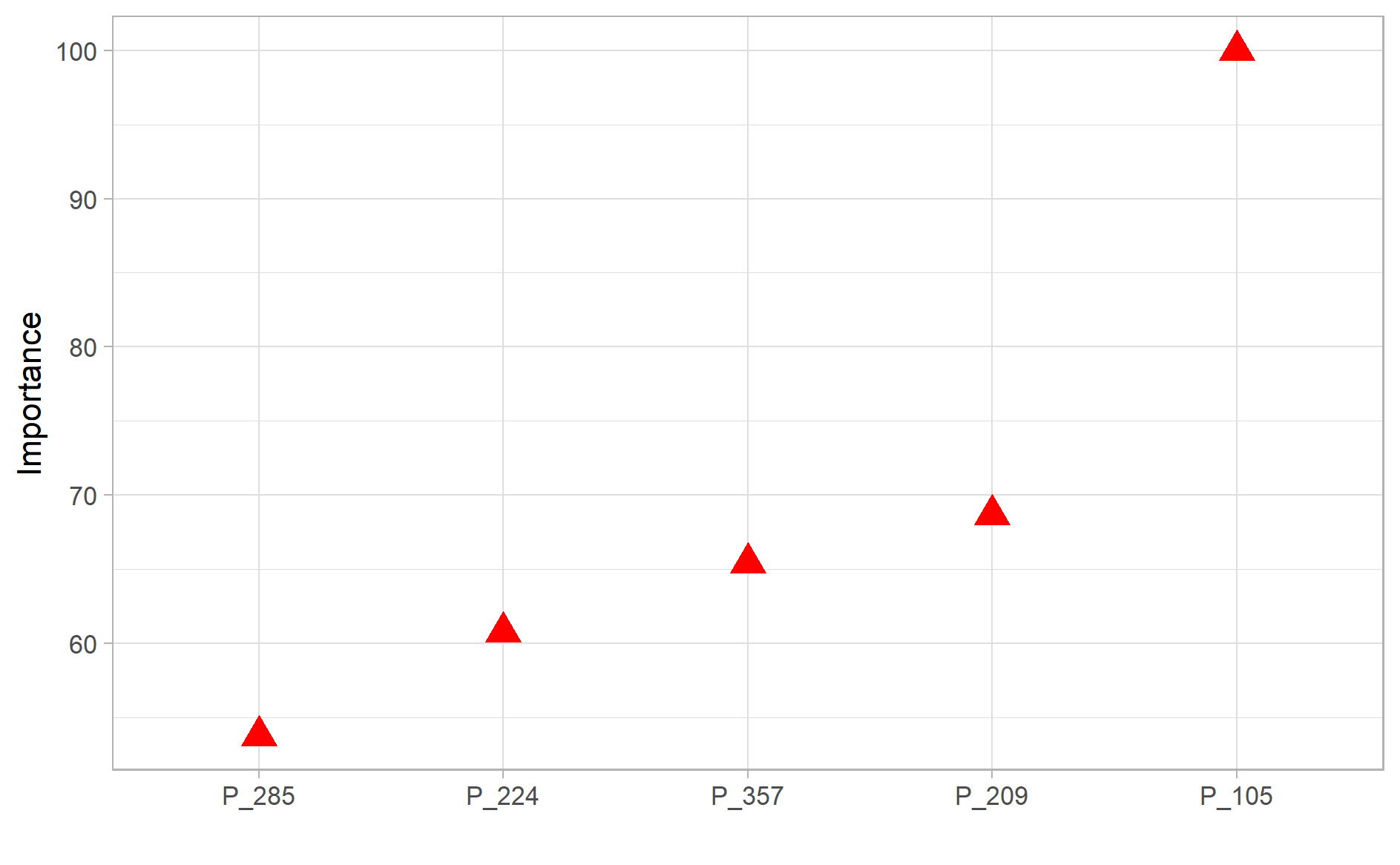


Figure E1: Features from Random Forest

## Section F: Simple GBM

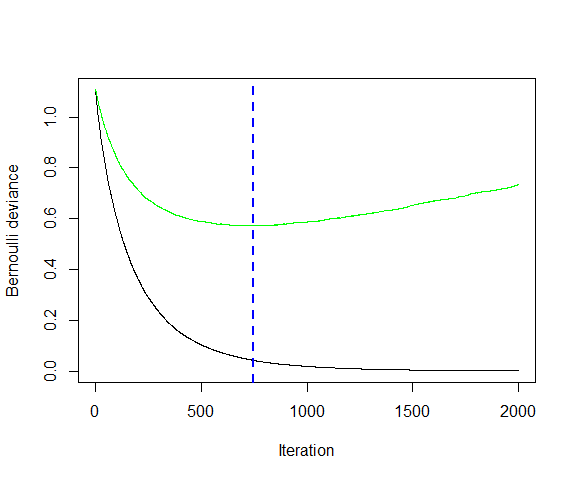


Figure F1: Bernoulli deviance Vs trees (test deviance Vs train deviance)

Table F1: GBM best model specification after cross validation

| shrinkage | interaction.depth | n.minobsinnode | n.trees | Accuracy | Kappa | AccuracySD | KappaSD |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0.3 | 6 | 20 | 400 | 0.9323892 | 0.7926398 | 0.0517419 | 0.1693404 |

Table F2: Confusion matrix and accuracy metrics for GMB after using optimum parameters.

| Train | Actual Control | Actual Alizahmer |
| --- | --- | --- |
| Control | 68 | 0 |
| Alizahmer | 0 | 213 |

|  | Train accuracy | Test Accuracy |
| --- | --- | --- |
| Accuracy | 1.00000 | 0.91597 |
| Sensitivity | 1.00000 | 0.94444 |
| Specificity | 1.00000 | 0.82759 |
| Pos Pred Value | 1.00000 | 0.94444 |
| Neg Pred Value | 1.00000 | 0.82759 |
| Prevalence | 0.75801 | 0.75630 |
| Detection Rate | 0.75801 | 0.71429 |

| Test | Actual Control | Actual Alizahmer |
| --- | --- | --- |
| Control | 24 | 5 |
| Alizahmer | 5 | 85 |

## 

## Section G: Stacking - custom tuned

For building custom base learners, example: for simple boosting (*gbm*), I used the following for tuning the parameters: interaction depth = 1 to 7; number of trees from 500 to 3,000; shrinkage = 0.5, 0.3, 0.1, 0.05, 0.01; and the minimum number of observations in a node of the tree 5, 10, 15, 20. For elastic net (*glmnet*) between 0 and 1 with 0.01 step and lambda were selected from hundred values that goes from ten to the power of 5 and ten to the power of -18. The code is provided under stacking-custom section.

Table G1: Correlation between the base learners model

|  | glmnet | gbm | knn |
| --- | --- | --- | --- |
| glmnet | 1.00 | 0.51 | 0.59 |
| gbm | 0.51 | 1.00 | 0.37 |
| knn | 0.59 | 0.37 | 1.00 |

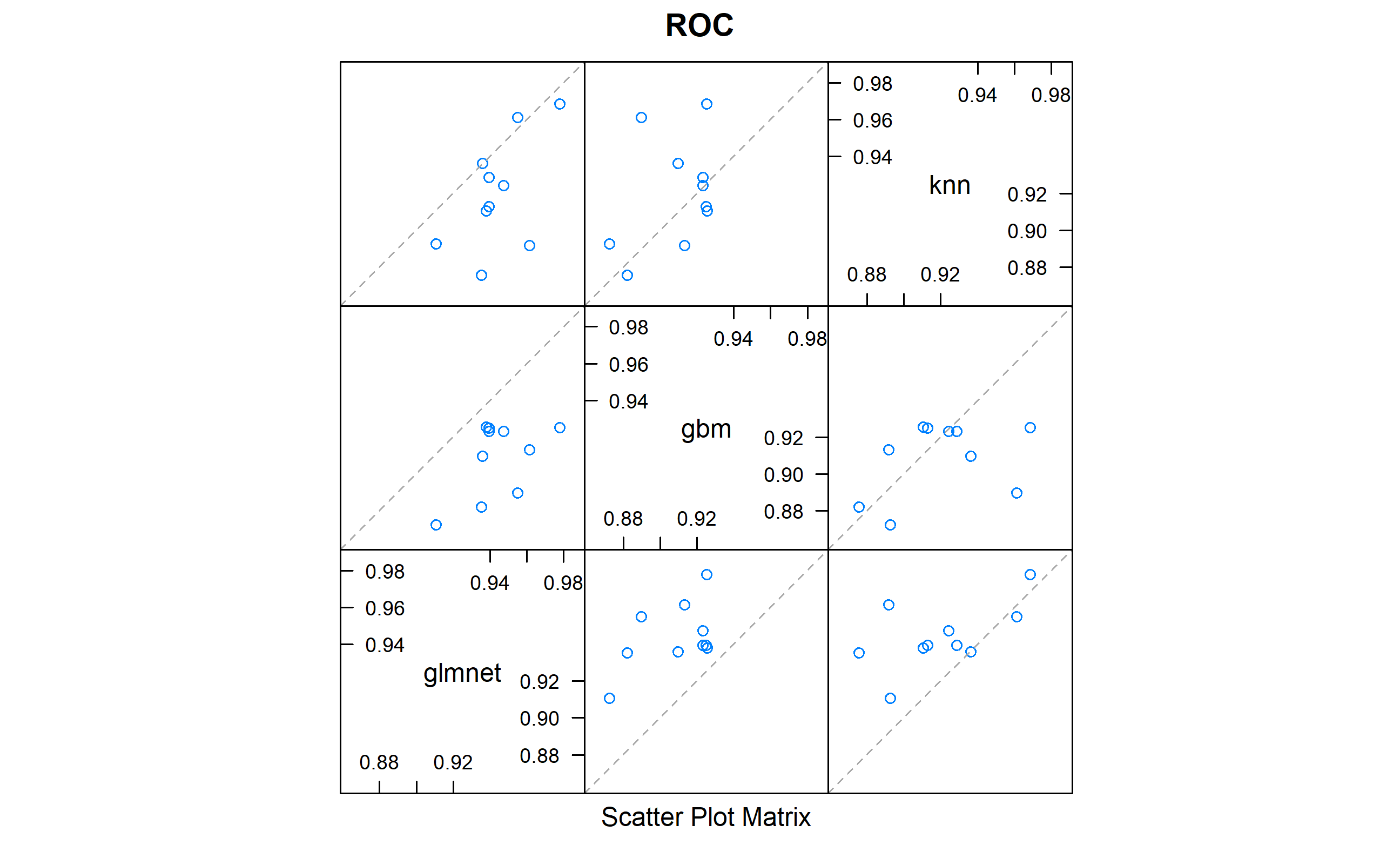


Figure G1: Correlation plot for the base learners

Table G2: Summary result for the stacked model (positive class was ‘Control’ so to compare it with other models here specificity represents sensitivity)

| term | class | estimate | conf.low | conf.high | p.value |
| --- | --- | --- | --- | --- | --- |
| accuracy | NA | 0.96 | 0.9 | 0.99 | 0 |
| kappa | NA | 0.88 | NA | NA | NA |
| mcnemar | NA | NA | NA | NA | 1 |
| sensitivity | Control | 0.93 | NA | NA | NA |
| specificity | Control | 0.97 | NA | NA | NA |
| pos\_pred\_value | Control | 0.90 | NA | NA | NA |
| neg\_pred\_value | Control | 0.98 | NA | NA | NA |
| precision | Control | 0.90 | NA | NA | NA |
| recall | Control | 0.93 | NA | NA | NA |
| f1 | Control | 0.91 | NA | NA | NA |
| prevalence | Control | 0.24 | NA | NA | NA |
| detection\_rate | Control | 0.22 | NA | NA | NA |
| detection\_prevalence | Control | 0.24 | NA | NA | NA |
| balanced\_accuracy | Control | 0.95 | NA | NA | NA |

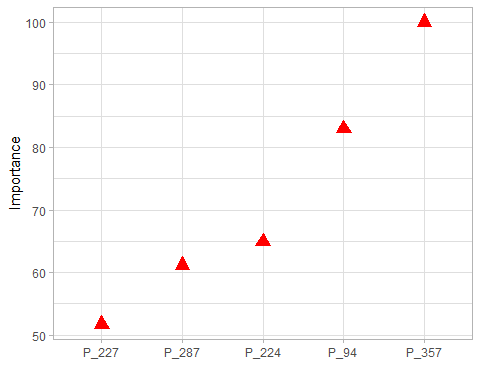


Figure G2: The first 5 predictors for the stacked model

## Section G: What else?

I have tried other models as well, autoencoders with logistic regression (test accuracy: 93%), autoencoders with well-tuned random forest (test accuracy 89%), auto-encoders with well-tuned XGBoost (test accuracy 91%), and PCA with logistic regression (test accuracy 97%). This also shows methods with linear properties outperform non-linear and other models.

## 

## Code Appendix

knitr::opts\_chunk$set(fig.pos = 'H')  
### Setting up the packages  
library(knitr)  
knitr::opts\_chunk$set(echo = FALSE)  
# check if packages are installed; if not, install them  
packages <- c("tidyverse", "readr", "ggExtra", "plotly",  
 "ggplot2","ggstatsplot","ggside","rigr","nlme","lmtest",  
 "sandwich","gridExtra","broom","praznik","tidyr","DMwR2",  
 "smotefamily","rpart","tree","kernlab","caret","glmnet",  
 "MASS","vip","caret","class","gbm")  
not\_installed <- setdiff(packages, rownames(installed.packages()))  
if (length(not\_installed)) install.packages(not\_installed)  
  
# load packages  
library(sandwich)  
library(gbm)  
library(randomForest)  
library(readr)  
library(lmtest)  
library(ggcorrplot)  
library(caret)  
library(vip)  
library(caretEnsemble)  
library(doParallel)  
library(MASS)  
library(nlme)  
library(tree)  
library(glmnet)  
library(broom)  
library(class)  
library(ggstatsplot)  
library(kernlab)  
library(ggside)  
library(rigr)  
library(rpart.plot)  
library(rpart)  
library(caret)  
library("DMwR2")  
library(pROC)  
library(praznik)  
library(ggExtra)  
library(gridExtra)  
library(plotly)  
library(ggplot2)  
library(tidyr)  
library(tidyverse)   
  
  
### -----------------------------------------------------------  
#Loading working directory of the raw data  
  
#Please load your data/directory by changing it with your work directory  
#Throughout this code module you will see a tone of places, where  
#data is read and written, so please make sure to change them to your  
#working directory folder format  
  
working\_directory\_data <- setwd("C:/Users/latera/Desktop/ML\_ass")  
  
#loads the data on a variable df  
load("data/ADProj.RData")  
  
study\_data <- Map(as.data.frame, ADProj)  
study\_train\_data <- study\_data[["X\_train"]]  
study\_train\_respose <- study\_data[["y\_train"]]  
  
  
study\_test\_data <- study\_data[["X\_test"]]  
study\_train\_final <- cbind(study\_train\_data,study\_train\_respose)  
  
study\_train\_final %>%  
 mutate(Outcome = factor(Outcome, levels = c("C", "AD"),   
 labels = c("Control", "Alizahmer"))) ->   
 study\_train\_final  
sum(is.na(study\_data))  
  
summary <- study\_train\_final %>%  
 group\_by(Outcome) %>%  
 summarise(observations = n())  
  
knitr::kable(summary, caption = "Summary table of the data")  
  
set.seed(2)  
study\_data\_idx = createDataPartition(study\_train\_final$Outcome,   
 p = 0.70, list = FALSE)  
study\_data\_trn = study\_train\_final[study\_data\_idx, ]  
study\_data\_tst = study\_train\_final[-study\_data\_idx, ]  
#Model for returning best performing model, if needed  
get\_best\_result = function(fit\_from\_caret) {  
 best\_model = which(rownames(fit\_from\_caret$results) ==  
 rownames(fit\_from\_caret$bestTune))  
 optimum\_result = fit\_from\_caret$results[best\_model, ]  
 rownames(optimum\_result) = NULL  
 optimum\_result  
}  
#Calculating accuracy  
calc\_acc = function(actual, predicted) {  
 mean(actual == predicted)  
}  
#Creating a Bayes classifier  
create\_bayes\_classifier <- function(model\_fit) {  
  
 function(data, thresholds) {  
  
 add\_class <- function(threshold) {  
  
 classes\_predicted <- factor(  
 model\_fit(data) > threshold,  
 levels = c(FALSE, TRUE),  
 labels = c("Control", "Alizahmer")  
 )  
  
 data %>% dplyr::mutate(  
 classes\_predicted = classes\_predicted,  
 threshold = threshold  
 )  
  
 }  
  
 thresholds %>% purrr::map\_dfr(add\_class)  
 }  
  
}  
set.seed(2)  
seat\_tree = rpart(Outcome ~ ., data = study\_data\_trn, method = "class")  
seat\_tree\_tst\_pred = predict(seat\_tree, study\_data\_tst, type = "class")  
seat\_tree\_trn\_pred = predict(seat\_tree, study\_data\_trn, type = "class")  
tst\_tab = table(predicted = seat\_tree\_tst\_pred, actual = study\_data\_tst$Outcome)  
trn\_tab = table(predicted = seat\_tree\_trn\_pred, actual = study\_data\_trn$Outcome)  
#Bayes Rule - test  
prob\_tst <- predict(seat\_tree, newdata = study\_data\_tst, type = "prob")  
predictions\_bayes\_tst <- ifelse(prob\_tst[,1] > 0.5, "Control", "Alizahmer")  
accuracy\_bayes\_tst <- calc\_acc(study\_data\_tst$Outcome, predictions\_bayes\_tst)  
print(paste("Accuracy (Bayes rule):", accuracy\_bayes\_tst))  
#Bayes Rule train  
prob\_trn <- predict(seat\_tree, study\_data\_trn, type = "prob")  
predictions\_bayes\_trn <- ifelse(prob\_trn[,1] > 0.5, "Alizahmer", "Control")  
accuracy\_bayes\_trn <- calc\_acc(study\_data\_trn$Outcome, predictions\_bayes\_trn)  
print(paste("Accuracy (Bayes rule):", accuracy\_bayes\_trn))  
trn\_con\_mat = confusionMatrix(trn\_tab, positive = "Alizahmer")  
  
knitr::kable(trn\_tab,   
 col.names = c("Control", "Alizahmer"),  
 digits = 5, caption = "Confussion matrix for test data")  
   
knitr::kable(c(trn\_con\_mat$overall["Accuracy"],  
trn\_con\_mat$byClass["Sensitivity"],  
trn\_con\_mat$byClass["Specificity"],   
trn\_con\_mat$byClass["Pos Pred Value"],  
trn\_con\_mat$byClass["Neg Pred Value"],  
trn\_con\_mat$byClass["Prevalence"],  
trn\_con\_mat$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for test data")  
  
tst\_con\_mat = confusionMatrix(tst\_tab, positive = "Alizahmer")  
  
knitr::kable(tst\_tab,   
 col.names = c("Control", "Alizahmer"),  
 digits = 5, caption = "Confussion matrix for test data")  
   
knitr::kable(c(tst\_con\_mat$overall["Accuracy"],  
tst\_con\_mat$byClass["Sensitivity"],  
tst\_con\_mat$byClass["Specificity"],   
tst\_con\_mat$byClass["Pos Pred Value"],  
tst\_con\_mat$byClass["Neg Pred Value"],  
tst\_con\_mat$byClass["Prevalence"],  
tst\_con\_mat$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for test data")  
  
min\_cp = seat\_tree$cptable[which.min(seat\_tree$cptable[,"xerror"]),"CP"]  
set.seed(2)  
cv\_tree <- train(  
 Outcome ~ .,  
 data = study\_data\_trn,  
 method = "rpart",  
 trControl = trainControl(method = "cv", number = 20),  
 tuneLength = 40  
)  
  
min\_params <- get\_best\_result(cv\_tree)  
ggplot(cv\_tree)+  
 theme\_bw() +  
 theme(axis.line = element\_line(colour = "white"),  
 panel.border = element\_blank(),  
 panel.background = element\_blank())+  
 theme(legend.title = element\_blank())  
seat\_rpart\_prune = prune(seat\_tree, cp = min\_params[1,1])  
seat\_prune\_trn\_pred <- predict(seat\_rpart\_prune, study\_data\_trn, type = "class")  
seat\_prune\_tst\_pred <- predict(seat\_rpart\_prune, study\_data\_tst, type = "class")  
seat\_prune\_trn\_pred = predict(seat\_rpart\_prune, study\_data\_trn, type = "class")  
  
trn\_con\_mat = confusionMatrix(table(predicted = seat\_prune\_trn\_pred, actual = study\_data\_trn$Outcome), positive = "Alizahmer")  
  
knitr::kable(table(predicted = seat\_prune\_trn\_pred, actual = study\_data\_trn$Outcome),   
 col.names = c("Control", "Alizahmer"),  
 digits = 5, caption = "Confussion matrix for test data")  
   
knitr::kable(c(trn\_con\_mat$overall["Accuracy"],  
trn\_con\_mat$byClass["Sensitivity"],  
trn\_con\_mat$byClass["Specificity"],   
trn\_con\_mat$byClass["Pos Pred Value"],  
trn\_con\_mat$byClass["Neg Pred Value"],  
trn\_con\_mat$byClass["Prevalence"],  
trn\_con\_mat$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for test data")  
seat\_prune\_tst\_pred = predict(seat\_rpart\_prune, study\_data\_tst, type = "class")  
  
tst\_con\_mat = confusionMatrix(table(predicted = seat\_prune\_tst\_pred, actual = study\_data\_tst$Outcome), positive = "Alizahmer")  
  
knitr::kable(table(predicted = seat\_prune\_tst\_pred, actual = study\_data\_tst$Outcome),   
 col.names = c("Control", "Alizahmer"),  
 digits = 5, caption = "Confussion matrix for test data")  
   
knitr::kable(c(tst\_con\_mat$overall["Accuracy"],  
tst\_con\_mat$byClass["Sensitivity"],  
tst\_con\_mat$byClass["Specificity"],   
tst\_con\_mat$byClass["Pos Pred Value"],  
tst\_con\_mat$byClass["Neg Pred Value"],  
tst\_con\_mat$byClass["Prevalence"],  
tst\_con\_mat$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for test data")  
  
set.seed(2)  
model\_glm = glm(Outcome ~ ., data = study\_data\_trn,  
 family = "binomial")  
  
log\_bayes\_pred <- create\_bayes\_classifier(  
 function(data) predict(model\_glm, data, type = "response")  
)  
train\_pred = log\_bayes\_pred(study\_data\_trn, 0.5)  
train\_tab\_log = table(train\_pred$classes\_predicted, study\_data\_trn$Outcome)  
train\_con\_mat\_log = confusionMatrix(train\_tab\_log, positive = "Alizahmer")  
trest\_pred = log\_bayes\_pred(study\_data\_tst, 0.5)  
test\_tab\_log = table(trest\_pred$classes\_predicted, study\_data\_tst$Outcome)  
test\_con\_mat\_log = confusionMatrix(test\_tab\_log, positive = "Alizahmer")  
set.seed(2)  
  
elastic\_mod = train(  
 Outcome ~ ., data = study\_data\_trn,  
 method = "glmnet",  
 trControl = trainControl(method = "cv", number = 5),  
 tuneLength = 20,  
 tuneGrid = expand.grid(alpha = seq(0, 1, by = 0.01),   
 lambda = 10^seq(5, -18, length = 100)),  
 family = "binomial"  
)  
  
knitr::kable(get\_best\_result(elastic\_mod), caption = "GLM net model specifications cross validation")  
  
pred\_bayes\_regu\_cv <- create\_bayes\_classifier (  
 function(data) predict(elastic\_mod, data, type="prob")[,"Alizahmer"]  
)  
train\_pred = pred\_bayes\_regu\_cv(study\_data\_trn, 0.5)  
train\_tab = table(train\_pred$classes\_predicted, study\_data\_trn$Outcome)  
train\_con\_mat\_log\_reg = confusionMatrix(train\_tab, positive = "Alizahmer")  
  
knitr::kable(train\_tab,   
 col.names = c("Actual Control", "Actual Alizahmer"),  
 digits = 5,caption = "Confussion matrix for train data")  
  
knitr::kable(c(train\_con\_mat\_log\_reg$overall["Accuracy"],  
train\_con\_mat\_log\_reg$byClass["Sensitivity"],  
train\_con\_mat\_log\_reg$byClass["Specificity"],   
train\_con\_mat\_log\_reg$byClass["Pos Pred Value"],  
train\_con\_mat\_log\_reg$byClass["Neg Pred Value"],  
train\_con\_mat\_log\_reg$byClass["Prevalence"],  
train\_con\_mat\_log\_reg$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for   
train data")  
tst\_pred = pred\_bayes\_regu\_cv(study\_data\_tst, 0.5)  
tst\_tab = table(tst\_pred$classes\_predicted, study\_data\_tst$Outcome)  
tst\_con\_mat\_log\_reg = confusionMatrix(tst\_tab, positive = "Alizahmer")  
  
knitr::kable(tst\_tab,   
 col.names = c("Actual Control", "Actual Alizahmer"),  
 digits = 5,caption = "Confussion matrix for train data")  
  
knitr::kable(c(tst\_con\_mat\_log\_reg$overall["Accuracy"],  
tst\_con\_mat\_log\_reg$byClass["Sensitivity"],  
tst\_con\_mat\_log\_reg$byClass["Specificity"],   
tst\_con\_mat\_log\_reg$byClass["Pos Pred Value"],  
tst\_con\_mat\_log\_reg$byClass["Neg Pred Value"],  
tst\_con\_mat\_log\_reg$byClass["Prevalence"],  
tst\_con\_mat\_log\_reg$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for   
train data")  
test\_prob = predict(elastic\_mod, newdata = study\_data\_tst,   
 type = "prob")[, "Alizahmer"]  
test\_roc = roc(study\_data\_tst$Outcome ~ test\_prob, plot = TRUE,   
 print.auc = TRUE, col=8)  
non\_zero <- data.frame(as.matrix(coef(elastic\_mod$finalModel, elastic\_mod$bestTune$lambda)))  
  
  
set.seed(2)  
confusion\_matrix <- function(data, classes\_ob, classes\_predicted) {  
  
 data %>%  
 dplyr::group\_by({{classes\_ob}}, {{classes\_predicted}}, .add = TRUE) %>%  
 dplyr::summarise(count = n()) %>%  
 tidyr::pivot\_wider(  
 names\_from = {{classes\_predicted}},  
 values\_from = count,  
 values\_fill = 0  
 ) %>%  
 dplyr::ungroup({{classes\_ob}}) %>%  
 dplyr::rename(`Predicted -> Observed` = {{classes\_ob}})  
  
}  
  
set.seed(2)  
prediction\_knn <- function(data, k\_neighbors) {  
 # Scale the features in the training data  
 df\_train\_scale <- study\_data\_trn %>%  
 mutate(across(P\_1:P\_360, ~scale(as.numeric(.x))))  
  
 # Scale the features in the test data  
 df\_test\_scale <- data %>%  
 mutate(across(P\_1:P\_360, ~scale(as.numeric(.x))))  
  
 # Function to add predictions based on a given number of nearest neighbors  
 add\_preds <- function(k) {  
 data %>%  
 mutate(  
 classes\_predicted = knn(  
 train = dplyr::select(df\_train\_scale, P\_1:P\_360),  
 test = dplyr::select(df\_test\_scale, P\_1:P\_360),  
 cl = df\_train\_scale$Outcome,  
 k = k  
 ),  
 k = k  
 )  
 }  
  
 # Apply the add\_preds function to each number of neighbors in k\_neighbors  
 k\_neighbors %>% map\_dfr(add\_preds)  
}  
  
classes <- list(  
 train = prediction\_knn(study\_data\_trn, c(1:20)) %>% group\_by(k),  
 test = prediction\_knn(study\_data\_tst, c(1:20)) %>% group\_by(k)  
)  
confusion\_matrix\_knn <- list(  
 train = classes$train %>% confusion\_matrix(Outcome, classes\_predicted),  
 test = classes$test %>% confusion\_matrix(Outcome, classes\_predicted)  
)  
accuracy\_knn <- list(  
 train = classes$train %>% summarise(accuracy = mean(Outcome == classes\_predicted)),  
 test = classes$test %>% summarise(accuracy = mean(Outcome ==  
 classes\_predicted))  
)  
set.seed(2)  
pred\_acc\_knn <-  
 list(train = study\_data\_trn, test = study\_data\_tst) %>%  
 purrr::map(~prediction\_knn(.x, 1:20) %>% group\_by(k)) %>%  
 purrr::map\_dfr(  
 ~summarise(.x, accuracy = mean(Outcome == classes\_predicted)),  
 .id = "data\_set"  
 )  
plot\_pred\_acc\_knn <-  
 ggplot(pred\_acc\_knn, aes(x = k, y = accuracy, color = data\_set)) +  
 geom\_line(lty = "dashed") +  
 geom\_point() +  
 theme\_bw() +  
 theme(axis.line = element\_line(colour = "white"),  
 axis.ticks = element\_blank(),  
 panel.grid.major = element\_blank(),  
 panel.grid.minor = element\_blank(),  
 panel.border = element\_blank(),  
 panel.background = element\_blank())+   
 theme(legend.background = element\_rect  
 (fill = "transparent"))+  
 labs(  
 title = "Predictive Accuracy - KNN",  
 x = "K values",  
 y = "Accuracy",  
 color = "Train/Test"  
 )  
plot(plot\_pred\_acc\_knn)  
set.seed(2)  
df\_train\_scale <- study\_data\_trn %>%  
 mutate(across(P\_1:P\_360, ~scale(as.numeric(.x))))  
  
# Scale the features in the test data  
df\_test\_scale <- study\_data\_tst %>%  
 mutate(across(P\_1:P\_360, ~scale(as.numeric(.x))))  
  
model\_knn <-knn(  
 train = dplyr::select(df\_train\_scale, P\_1:P\_360),  
 test = dplyr::select(df\_test\_scale, P\_1:P\_360),  
 cl = df\_train\_scale$Outcome,  
 k = 12)  
  
set.seed(2)  
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)  
  
knn\_fit <- train(Outcome ~., data = study\_data\_trn, method = "knn",  
 trControl=trctrl,  
 preProcess = c("center", "scale"),  
 tuneLength = 20)  
pred\_bayes\_knn\_cv <- create\_bayes\_classifier (  
 function(data) predict(knn\_fit, data, type="prob")[,"Alizahmer"]  
)  
train\_pred = pred\_bayes\_knn\_cv(study\_data\_trn, 0.5)  
train\_tab = table(train\_pred$classes\_predicted, study\_data\_trn$Outcome)  
train\_con\_mat\_knn\_reg = confusionMatrix(train\_tab, positive = "Alizahmer")  
  
knitr::kable(train\_tab,   
 col.names = c("Actual Control", "Actual Alizahmer"),  
 digits = 5,caption = "Confussion matrix for train data")  
  
knitr::kable(c(train\_con\_mat\_knn\_reg$overall["Accuracy"],  
train\_con\_mat\_knn\_reg$byClass["Sensitivity"],  
train\_con\_mat\_knn\_reg$byClass["Specificity"],   
train\_con\_mat\_knn\_reg$byClass["Pos Pred Value"],  
train\_con\_mat\_knn\_reg$byClass["Neg Pred Value"],  
train\_con\_mat\_knn\_reg$byClass["Prevalence"],  
train\_con\_mat\_knn\_reg$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for   
train data")  
tst\_pred = pred\_bayes\_knn\_cv(study\_data\_tst, 0.5)  
tst\_tab = table(tst\_pred$classes\_predicted, study\_data\_tst$Outcome)  
tst\_con\_mat\_knn\_reg = confusionMatrix(tst\_tab, positive = "Alizahmer")  
  
knitr::kable(tst\_tab,   
 col.names = c("Actual Control", "Actual Alizahmer"),  
 digits = 5,caption = "Confussion matrix for train data")  
  
knitr::kable(c(tst\_con\_mat\_knn\_reg$overall["Accuracy"],  
tst\_con\_mat\_knn\_reg$byClass["Sensitivity"],  
tst\_con\_mat\_knn\_reg$byClass["Specificity"],   
tst\_con\_mat\_knn\_reg$byClass["Pos Pred Value"],  
tst\_con\_mat\_knn\_reg$byClass["Neg Pred Value"],  
tst\_con\_mat\_knn\_reg$byClass["Prevalence"],  
tst\_con\_mat\_knn\_reg$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for   
train data")  
set.seed(2)   
bagged\_trees\_rf <- randomForest(Outcome ~ ., data = study\_data\_trn, mtry = sqrt(ncol(study\_data\_trn) - 1), importance=TRUE)  
train\_pred\_rf <- predict(bagged\_trees\_rf, study\_data\_trn)  
test\_pred\_rf <- predict(bagged\_trees\_rf, study\_data\_tst)  
set.seed(2)  
  
control <- trainControl(method='repeatedcv',   
 number=10,   
 repeats=3,   
 search='grid')  
  
tunegrid <- expand.grid(.mtry = 360 \* c(.05, .15, .25, .333, .4, 0.5, 1))   
  
rf\_gridsearch <- train(Outcome ~ .,   
 data = study\_data\_trn,  
 method = 'rf',  
 metric = 'Accuracy',  
 tuneGrid = tunegrid)  
plot(rf\_gridsearch)  
train\_pred\_rf <- predict(rf\_gridsearch, study\_data\_trn)  
test\_pred\_rf <- predict(rf\_gridsearch, study\_data\_tst)  
  
stu\_train\_int <- study\_data\_trn %>% mutate(Outcome = as.integer(Outcome) - 1L)  
stu\_test\_int <- study\_data\_tst %>% mutate(Outcome = as.integer(Outcome) - 1L)  
  
  
set.seed(2)  
boost\_model <- gbm(Outcome ~ ., data = stu\_train\_int, distribution = "bernoulli",  
 n.trees = 2000, interaction.depth = 6, shrinkage = 0.01,  
 cv.folds = 10)  
  
# Compute the predicted labels for the training and test data  
train\_preds\_boost\_no\_cv <- ifelse(predict(boost\_model, study\_data\_trn, type = "response") > 0.5, "Alizahmer", "Control")  
test\_preds\_boost\_no\_cv <- ifelse(predict(boost\_model, study\_data\_tst, type =   
 "response") > 0.5, "Alizahmer", "Control")  
  
  
#Parallel computing with 7 workers  
#cl <- makePSOCKcluster(7)  
#registerDoParallel(cl)  
  
set.seed(2)  
cv\_10 = trainControl(method = "cv", number = 10)  
  
gbm\_grid = expand.grid(*interaction.depth* = 1:7,  
 n.trees = (1:6) \* 500,  
 s*hrinkage* = c(0.5, 0.3, 0.1, 0.05, 0.01),  
 *n.minobsinnode* = c(5, 10, 15, 20))

seat\_gbm\_tune = train(Outcome ~ ., data = study\_data\_trn,  
 method = "gbm",  
 trControl = cv\_10,  
 verbose = FALSE,  
 tuneGrid = gbm\_grid)  
  
knitr::include\_graphics("CV\_boosting.png")  
  
set.seed(2)  
  
methods = c("glmnet", "gbm","knn")  
  
tc = trainControl(method = "repeatedcv", number = 10, repeats = 10, search = "grid", savePredictions = "final", index = createResample(study\_data\_trn$Outcome, 10), summaryFunction = twoClassSummary, classProbs = TRUE, verboseIter = TRUE)  
  
models = caretList(Outcome~., data = study\_data\_trn,   
 trControl = tc, methodList = methods)  
  
output = resamples(models)  
  
stack = caretStack(models, method="glmnet", trControl = tc)  
  
  
pred = predict(stack, study\_data\_tst)  
cm = confusionMatrix(study\_data\_tst$Outcome, pred)  
  
  
dotplot(output)  
  
test\_prob = predict(elastic\_mod, newdata = study\_test\_data)  
write.table(test\_prob, file = "new\_elastic\_97.txt", sep = "\t",  
 row.names = FALSE)  
  
specie <- c(rep("Outcome" , 2) )  
condition <- rep(c("Control", "Alizahmer") , 2)  
value = c(10,100)  
data <- data.frame(specie,condition,value)  
   
# Stacked  
ggplot(data, aes(fill=condition, y=value, x=specie)) +   
 geom\_bar(position="fill", stat="identity")+  
 xlab("Outcome type") + ylab("Percentage contribution")+theme\_minimal() +  
 theme(plot.background = element\_blank(),  
 panel.grid.minor = element\_blank(),  
 panel.grid.major.y = element\_blank(),  
 panel.grid.major.x = element\_blank()  
 )  
  
featurePlot(x = study\_data\_trn[, 10:13],  
y = study\_data\_trn$Outcome,  
plot = "pairs",  
auto.key = list(columns = 2))  
library("factoextra")  
library("ggplot2")  
  
pca\_study <- study\_train\_final  
  
  
  
# Extract predictor variables and center and scale them  
predictor\_vars <- pca\_study[, 1:360]  
  
# Compute principal components  
pca\_result <- prcomp(predictor\_vars)  
  
# Extract first two principal components  
pc1 <- pca\_result$x[, 1]  
pc2 <- pca\_result$x[, 2]  
  
ggplot(data = pca\_study, aes(x = pc1, y = pc2, color = Outcome)) +  
 geom\_point() +  
 xlab("Reduced dimention 1") + ylab("Reduced dimention 2")+theme\_minimal() +  
 theme(plot.background = element\_blank(),  
 panel.grid.minor = element\_blank(),  
 panel.grid.major.y = element\_blank(),  
 panel.grid.major.x = element\_blank()  
 )  
  
compute\_corr\_matrix <- function(df) {  
 # Select all columns except the first one (assumed to be the response variable)  
 X\_cols <- 1:360  
   
 # Compute the correlation matrix on the predictors  
 corr\_mat <- cor(df[, X\_cols])  
   
 # Plot the correlation matrix using ggcorrplot  
 ggcorrplot(corr\_mat, type = "lower", lab = FALSE,   
 title = "Correlation Matrix of Predictors",  
 ggtheme = ggplot2::theme\_gray,  
 legend.title = "Correlation scale")  
}  
  
compute\_corr\_matrix(study\_data\_trn)  
set.seed(2)  
#Un proned  
seat\_tree\_all = rpart(Outcome ~ ., data = study\_train\_final)  
#summary(seat\_tree)  
rpart.plot(seat\_tree\_all)  
title(main = "Unpruned Classification Tree - total data")  
rpart.plot(seat\_tree)  
title(main = "Unpruned Classification Tree - train")  
plotcp(seat\_tree)  
seat\_rpart\_prune = prune(seat\_tree, cp = min\_params[1,1])  
  
rpart.plot(seat\_rpart\_prune)  
  
title(main = "Pruned Classification Tree")  
  
vip(cv\_tree, num\_features = 40, bar = FALSE)  
model\_glm\_summary <- summary(model\_glm, correlation = TRUE)  
  
knitr::kable(  
 coef(model\_glm\_summary)[0:10,],  
 digits = 2,  
 caption = "Logistic regression training coefficients (for the first 10   
 predictors)"  
)  
  
knitr::kable(train\_tab\_log,   
 col.names = c("Actual Benign", "Actual Malignant"),  
 digits = 5,caption = "Confussion matrix for train data")  
  
knitr::kable(c(train\_con\_mat\_log$overall["Accuracy"],  
train\_con\_mat\_log$byClass["Sensitivity"],  
train\_con\_mat\_log$byClass["Specificity"],   
train\_con\_mat\_log$byClass["Pos Pred Value"],  
train\_con\_mat\_log$byClass["Neg Pred Value"],  
train\_con\_mat\_log$byClass["Prevalence"],  
train\_con\_mat\_log$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for   
train data")  
  
knitr::kable(test\_tab\_log,   
 col.names = c("Actual Benign", "Actual Malignant"),  
 digits = 5,caption = "Confussion matrix for test data")  
  
knitr::kable(c(test\_con\_mat\_log$overall["Accuracy"],  
test\_con\_mat\_log$byClass["Sensitivity"],  
test\_con\_mat\_log$byClass["Specificity"],   
test\_con\_mat\_log$byClass["Pos Pred Value"],  
test\_con\_mat\_log$byClass["Neg Pred Value"],  
test\_con\_mat\_log$byClass["Prevalence"],  
test\_con\_mat\_log$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for   
test data")  
plot(elastic\_mod)  
vip(elastic\_mod, num\_features = 20, geom = "point")  
X\_study\_trn <- study\_data\_trn %>% select(-Outcome) %>% as.matrix()  
X\_study\_tst <- study\_data\_tst %>% select(-Outcome) %>% as.matrix()  
  
Y\_study\_trn <- study\_data\_trn$Outcome  
Y\_study\_tst <- study\_data\_tst$Outcome  
# Create a grid of values for lambda  
lambda\_seq <- 10^seq(5, -18, length = 100)  
set.seed(2)  
# Fit the ridge logistic regression model  
ridge\_fit <- glmnet(X\_study\_trn, Y\_study\_trn, family = "binomial", alpha = 0,   
 lambda = lambda\_seq)  
  
coef\_P1 <- coef(ridge\_fit)["P\_1", ]  
  
  
# Plot the coefficients in function of log(lambda)  
plot(log(ridge\_fit$lambda), coef\_P1, type = "l", xlab = "log(lambda)",   
 ylab = "Coefficient")  
for (k in 2:360) {  
 coef\_P3 <- coef(ridge\_fit)[k, ]  
 lines(log(ridge\_fit$lambda), coef\_P3, type = "l", col = "blue")  
legend("bottomright", legend = c("Predictor 1 beta", "Predictor 2 beta"),   
 col = c("black", "red"), lty = 1)  
}  
  
abline(v=-10, col = "lightgray", lty = 3)  
# Fit the ridge logistic regression model  
cvfit <- cv.glmnet(X\_study\_trn, Y\_study\_trn, family = "binomial",   
 type.measure = "class",  
 alpha = 0, lambda = lambda\_seq)  
  
# Find optimal lambda value that minimizes CV error  
opt\_lambda <- cvfit$lambda.min  
#cat("Optimal lambda value:", opt\_lambda, "\n")  
  
  
# Plot mis-classification error vs. log(lambda)  
plot(cvfit, xlab = "Log(lambda)")  
print\_table <- function(k, data, type) {  
 print(knitr::kable(  
 x = data,  
 caption = sprintf("%s kNN confusion matrix (k = %d)", type, k)  
 ))  
}  
confusion\_matrix\_knn$train %>%  
 tidyr::nest(data = -k) %>%  
 purrr::pwalk(print\_table, type = "Training")  
confusion\_matrix\_knn$test %>%  
 tidyr::nest(data = -k) %>%  
 purrr::pwalk(print\_table, type = "Test")  
  
left\_join\_result <- left\_join(accuracy\_knn$train, accuracy\_knn$test, by = "k")  
  
knitr::kable(left\_join\_result,   
 caption = "Predictive accuracy - kNN",   
 col.names = c("k", "training", "test"),   
 digits = 3)  
plot(c(1:20), as.vector(1 - (accuracy\_knn$test[,'accuracy'])$accuracy), type = "b", col = "dodgerblue", cex = 1, pch = 20,   
 xlab = "k, number of neighbors", ylab = "classification error",  
 main = "(Test) Error Rate vs Neighbors")  
# add line for min error seen  
abline(h = min(as.vector(1 - (accuracy\_knn$test[,'accuracy'])$accuracy)), col = "darkorange", lty = 3)  
# add line for minority prevalence in test set  
abline(h = mean(study\_data\_tst == "Alizahmer"), col = "grey", lty = 2)  
plot(knn\_fit)  
knitr::kable(get\_best\_result(knn\_fit), caption = "kNN model specifications cross validation")  
train\_pred = predict(rf\_gridsearch, study\_data\_trn)  
train\_tab = table(train\_pred, study\_data\_trn$Outcome)  
train\_con\_mat\_knn\_reg = confusionMatrix(train\_tab, positive = "Alizahmer")  
  
knitr::kable(train\_tab,   
 col.names = c("Actual Control", "Actual Alizahmer"),  
 digits = 5,caption = "Confussion matrix for train data")  
  
knitr::kable(c(train\_con\_mat\_knn\_reg$overall["Accuracy"],  
train\_con\_mat\_knn\_reg$byClass["Sensitivity"],  
train\_con\_mat\_knn\_reg$byClass["Specificity"],   
train\_con\_mat\_knn\_reg$byClass["Pos Pred Value"],  
train\_con\_mat\_knn\_reg$byClass["Neg Pred Value"],  
train\_con\_mat\_knn\_reg$byClass["Prevalence"],  
train\_con\_mat\_knn\_reg$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for   
train data, random forest")  
test\_pred = predict(rf\_gridsearch, study\_data\_tst)  
test\_tab = table(test\_pred, study\_data\_tst$Outcome)  
train\_con\_mat\_rf = confusionMatrix(test\_tab, positive = "Alizahmer")  
  
knitr::kable(test\_tab,   
 col.names = c("Actual Control", "Actual Alizahmer"),  
 digits = 5,caption = "Confussion matrix for train data, random forest")  
  
knitr::kable(c(train\_con\_mat\_rf$overall["Accuracy"],  
train\_con\_mat\_rf$byClass["Sensitivity"],  
train\_con\_mat\_rf$byClass["Specificity"],   
train\_con\_mat\_rf$byClass["Pos Pred Value"],  
train\_con\_mat\_rf$byClass["Neg Pred Value"],  
train\_con\_mat\_rf$byClass["Prevalence"],  
train\_con\_mat\_rf$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for   
train data, random forest")  
  
vip(rf\_gridsearch, num\_features = 5, geom = "point", horizontal = FALSE,   
 aesthetics = list(color = "red", shape = 17, size = 4)) +  
 theme\_light()  
gbm.perf(boost\_model, method = "cv")  
knitr::kable(get\_best\_result(seat\_gbm\_tune), caption = "GBM model specifications cross validation")  
pred\_bayes\_boost\_cv <- create\_bayes\_classifier (  
 function(data) predict(seat\_gbm\_tune, data, type="prob")[,"Alizahmer"]  
)  
train\_pred = pred\_bayes\_boost\_cv(study\_data\_trn, 0.5)  
train\_tab = table(train\_pred$classes\_predicted, study\_data\_trn$Outcome)  
train\_con\_mat\_boost\_reg = confusionMatrix(train\_tab, positive = "Alizahmer")  
  
knitr::kable(train\_tab,   
 col.names = c("Actual Control", "Actual Alizahmer"),  
 digits = 5,caption = "Confussion matrix for train data")  
  
knitr::kable(c(train\_con\_mat\_boost\_reg$overall["Accuracy"],  
train\_con\_mat\_boost\_reg$byClass["Sensitivity"],  
train\_con\_mat\_boost\_reg$byClass["Specificity"],   
train\_con\_mat\_boost\_reg$byClass["Pos Pred Value"],  
train\_con\_mat\_boost\_reg$byClass["Neg Pred Value"],  
train\_con\_mat\_boost\_reg$byClass["Prevalence"],  
train\_con\_mat\_boost\_reg$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for   
train data")  
test\_pred = pred\_bayes\_boost\_cv(study\_data\_tst, 0.5)  
test\_tab = table(test\_pred$classes\_predicted, study\_data\_tst$Outcome)  
tst\_con\_mat\_boost\_reg = confusionMatrix(test\_tab, positive = "Alizahmer")  
  
knitr::kable(test\_tab,   
 col.names = c("Actual Control", "Actual Alizahmer"),  
 digits = 5,caption = "Confussion matrix for train data")  
  
knitr::kable(c(tst\_con\_mat\_boost\_reg$overall["Accuracy"],  
tst\_con\_mat\_boost\_reg$byClass["Sensitivity"],  
tst\_con\_mat\_boost\_reg$byClass["Specificity"],   
tst\_con\_mat\_boost\_reg$byClass["Pos Pred Value"],  
tst\_con\_mat\_boost\_reg$byClass["Neg Pred Value"],  
tst\_con\_mat\_boost\_reg$byClass["Prevalence"],  
tst\_con\_mat\_boost\_reg$byClass["Detection Rate"]),  
 col.names = c("Percentages"),  
 digits = 5,caption = "Confussion matrix for   
train data")  
  
# stacking-custom, section   
  
# The result of the following code is not included in this report.  
  
gbm\_grid\_2 = expand.grid(interaction.depth = 1:7,  
 n.trees = (1:6) \* 500,  
 shrinkage = c(0.5, 0.3, 0.1, 0.05, 0.01),  
 n.minobsinnode = c(5, 10, 15, 20))  
  
ensemble\_control <- caret::trainControl(  
 method="cv",  
 number=5,  
 verboseIter = FALSE,  
 savePredictions = "final")  
  
model\_list\_big <- caretEnsemble::caretList(Outcome~., data=study\_data\_trn,  
 trControl=ensemble\_control,  
tuneList=list(  
 elastic = caretEnsemble::caretModelSpec(method = "glmnet", family= "binomial",  
 tuneGrid = expand.grid(alpha = seq(0, 1, by = 0.01), lambda = lambda\_seq)),  
 xgb=caretEnsemble::caretModelSpec(method="gbm", tuneGrid=gbm\_grid\_2)),  
  
 metric="Accuracy",   
 methodList = c("glmnet", "gbm","knn")); model\_list\_big  
  
# correlation between results  
knitr::kable(modelCor(output), caption = "Correlation between the base learners model", digits = 2)  
  
splom(output)  
knitr::kable(tidy(cm), caption = "Summary result for the stacked model", digits = 2)  
  
vip(seat\_gbm\_tune, num\_features = 5, geom = "point", horizontal = FALSE,   
 aesthetics = list(color = "red", shape = 17, size = 4)) +  
 theme\_light()