Stat432 Final Project

2024-04-22

Contributers: "Aditya Kakarla(kakarla4), Sihun Wang(sihunw2), Carlos Bautista(cbauti20), Cristian Romaniz(romanis3)"

Literature Review

Dataset Background: The dataset to be reviewed, analyzed, and utilized for the remainder of this project originates from the Computational Intelligence and Learning (CoIL) challenge of the year 2000. This dataset was created using authentic data provided by a Dutch data mining company, Sentient Machine Research, and is divided into two parts.

The first part of the dataset is comprised of 5,822 customer records intended for training and validating prediction models. Each customer record contains 86 attributes. The following attributes are divided into two categories: sociodemographic data based on zip codes (religion, marital status, income, etc.) and product ownership (contribution and number of certain policies). It is important to note that customers that share the same zip code share identical socio-demographic attributes. In addition, the target variables are included as one of the attributes in this first part.

The second part of the dataset contains 4,000 customer records designed for predictions. This second part of the dataset follows the same format as the first dataset but does not include the target variables. Instead, the target variables for the evaluation set are provided separately, allowing for predictions and tests for accuracy to be made.

Previous Reports: The CoIL challenge asked participants to predict which customers would be interested in purchasing a caravan insurance policy, along with an explanation supported by evidence. Participants were expected to utilize the specified dataset and fit it through certain classification algorithms of their choice to create arguable predictions. Below are provided examples of specific classification algorithms previously used by participants, along with their approaches and the conclusions drawn from their predictions. In this following report, White et al. (2000) utilized the given dataset to develop a Binary Classification Tree model. This model is designed to split data into partitions, which will continue to break into additional branches (nodes) until the desired classifications are achieved. White et al. (2000) performed several data adjustments to the dataset. The data was carefully adjusted by recoding attributes to create a quasi-ordering, improving the efficiency of the classification process. Subsequently, White et al. (2000) performed twelve Bootstrap Aggregation (bagging) replicates of the training set, each with its own binary classification tree. Bagging, an ensemble method, involves taking multiple (replicated) samples from the training dataset and averaging their predictions to enhance the accuracy of the machine learning algorithm. The predictions from each tree were aggregated to determine the final model. This process confirmed a prediction score of 110 correct classifications, exceeding the expected 105 correct classifications without utilizing the bagging technique. White et al. (2000) concluded that the approaches they made enhanced the model's ability to predict correctly, concluding that the combinations of the methods used in their report improved the effectiveness and accuracy of their classification model.

In another report, Keerthi et al. (2000) tackled this challenge by using Support Vector Machines (SVM's). SVM's find the most optimal boundary (hyperplane) in a dimensional space that distinctly separates (classifies) data point of different classes. Additionally, in situations where the data is not linearly separable, Kernel

functions are used to transform the data points into a higher dimensional space, enabling SVM's to find a boundary (hyperplane) between data points. Keerthi et al. (2000) began their approach by removing less impactful sociodemographic attributes to focus more effectively on the target class. They encoded the categorical attributes, transforming them into multiple binary attributes, which allowed the SVM's to effectively categorize the data better. They experimented with various SVM kernels (Gaussian, Linear, Polynomial) and optimized parameters through 10-fold cross-validation. The SVM models were then evaluated using the hit rate (an evaluation metric used to assess the effectiveness of classification models). Initially, the approach involved using all 85 attributes, resulting in a hit rate of approximately 16%. After refining the dataset by focusing on more relevant attributes, the hit rates varied between 13.25% and 14.38%. Although these rates are lower, they demonstrated a more targeted and efficient model configuration. Keerthi et al. (2000) concluded that careful selection of attributes and careful data preparation are crucial for enhancing the accuracy of SVM classification models.

In conclusion, the following CoIL challenge reports by White et al. (2000) and Keerthi et al. (2000) demonstrated the application of distinct classification algorithms and approaches to predict which customers would be most likely interested in purchasing a caravan insurance policy. White et al. (2000) conducted a Binary Classification Tree model and incorporated Bootstrap Aggregation, indicating that combining these methods can significantly improve the prediction accuracy. In contrast, Keerthi et al. used a Support Vector Machines model and demonstrated the importance of using various kernel functions and focusing on parameter tuning could also improve accuracy in predictive analysis. Together, both reports highlight the significance of methodological adjustments in model configuration and data handling to effectively boost the accuracy of predictive modeling in an insurance context.

Our Goal and Approach:

Similar to the CoIL challenge, our goal is to predict which customers are most likely to purchase a caravan insurance policy. Additionally, we will enhance our understanding of the dataset we will be working with by summarizing statistics, creating data visualizations, and offering detailed explanations to deepen our knowledge of the dataset. We will develop four classification models in R Studio: Logistics Regression, K-Nearest-Neigbours, Supper Vector Machines and Decision Tree Analysis. We will attempt to use cross-validation, which is technique that is used for evaluating and improving the performance of machine learning models in order to reduce over fitting and there generalizibility on new, unseen data, in order to find the hyperparamter configurations, which are parameters that specify the details of the learning process, that optimize the performance of each classification model. Throughout our report, we will explain the significance of each classification model, describe our distinct model configurations and how we used cross-validation - this will be done through text and graphs. We will conclude by conducting a comparative analysis of error metrics, such as sensitivity, specificity, error rate, accuracy, precision and the F-1 score of the each of the models and then idenfiyingn which one is the best in a general sense and in the context of our dataset.

Through conducing this mutlifaced research paper where we implement a range of machine learning classification models and then analyze their performance not only will we able to build upon the concepts we learnt from class but are able to obtain a deeper understanding of the nuances of the functionalities of several machine learning algorithims

Exploratory Data Analysis and Summary Statistics

The dimensions of the traning and then testing set are shown below

From the dimensions of the data set we can see that there are 86 predictor variables. Due to the extensive number of predictor, variables, it is central to implement dimension reduction techniques and feature engineering in order to reduce the number of predictor variables that we fit in the final model. A model with too many predictors can become unnecessarily complex which which can often lead to overfitting, which is the phenomenon in which the model ends up fitting well to the training data, but in that process in fits the noise in the training data as well rather than the underlying trend in the data, which eventually leads to

poor generalization on new unseen data - in other words the model would have low bias and high variance. Moreover, an exceedingly complex model can lead to a phenomenon in which the model is hard to interpret and understand, which makes deciphering final conclusions harder - for instance it would be harder to understand which feature variables are more important in explaining the variance of the response variable. In addition to this, a higher number if predictors can often mean that the likelihood of the predictors being correlated also increases, in other words the likelihood and amount of multidisciplinary increases - this can make deciphiring the effect of the each predictor variable on the response variables harder. Lastly, more predictors also increases the computational compexity an d the data requirements of the model

```
## Number of integer variables in the dataset: 86
```

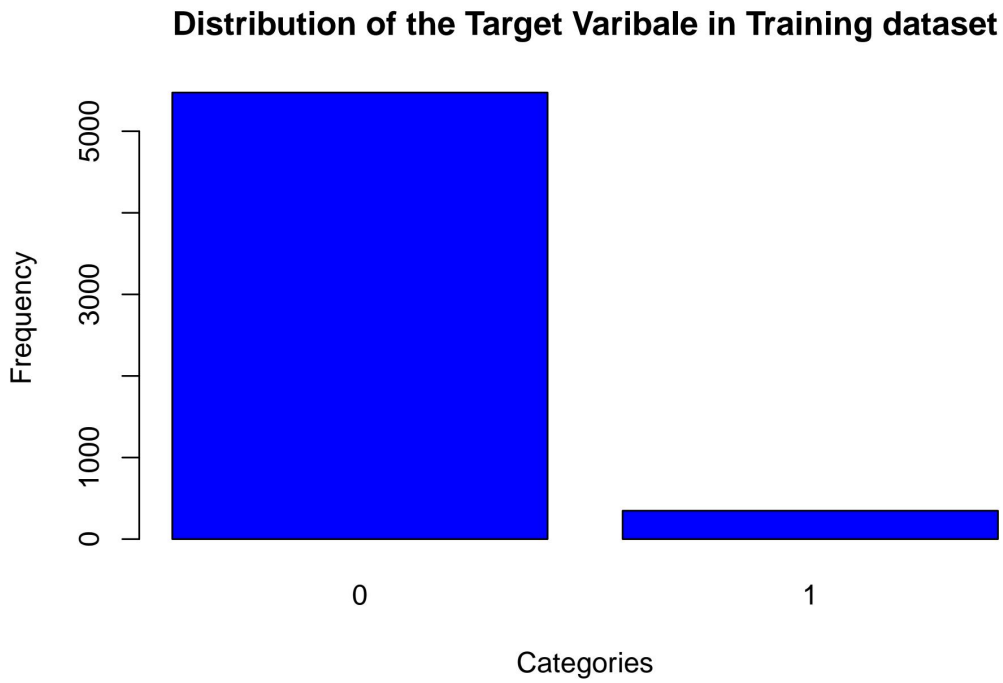
```
## Number of character variables in the dataset: 0
```

One of the most important parts of datapreprocessing is cleaning the data to check for any missing values. However we can see that niether the training or testing dataset have any missing values therefore implement techniques such as imputation to deal such a problem.

```
## The number of missing values in the training dataset is 0
```

```
## The number of missing values in the testing dataset is 0
```

The graph below depicts the disruption in the categorizes for our response variable. We can see that there are two class of 0 and 1 which correspond to no and yes for having a caravan insurance policy. The actual counts



are depicted below that.

```r
table(training_df$V86)
```

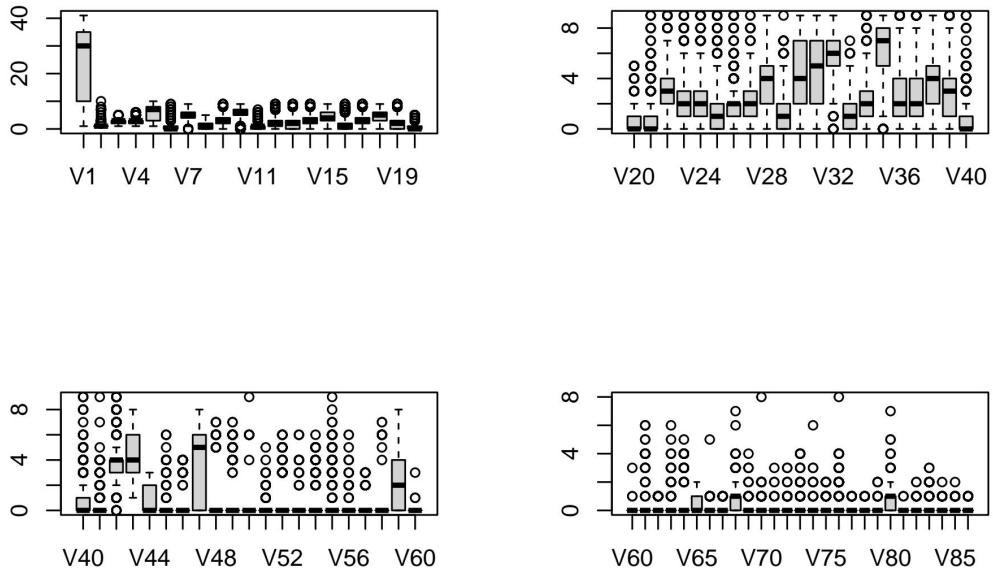
```
##
##    0    1
## 5474  348
```

From the output above we can clearly see that one class(class 0) is heavily over presented in comparison to class 1. This could create an issue of model bias where the machine learning models we implement willtend to favor the majority class. This may create issues with the accuracy of the model and overfitting as the models may perform well overal, however they simultaneously fail to caputre the range of characteristics and patterns of the minority class. Therefore we will implement under sampling in order to reduce the number of instances from the class that is underrepresented and create a more balanced dataset. We only apply the undersampling to the training dataset in order to deal with the issue of imbalanced data. It is important that we choose the right value of N, which is the total number of observations we have in our dataset after the sampling process is completed. We select a value of N that ensure that we based on the idea of creating a more balanced data while being able to retain the as much information and data from the original dataset as possible

```r
training_df_sampled <- ovun.sample(V86 ~ ., data = training_df, method = "under", N = 3500, seed = 1)$da
table(training_df_sampled$V86)
```

```
##
##    0    1
## 3152  348
```

Another way to clean our dataset is to find outliers, this is important to ensure that there are no inaccuaracies in our data. From the boxplot below we can see that there are many outliers in our dataset. We will use - the z-score method to remove all of the outliers, however due to the large number of predictors we have we have decided not to remove outliers.



Data Preprocessing: Principal Component Analysis

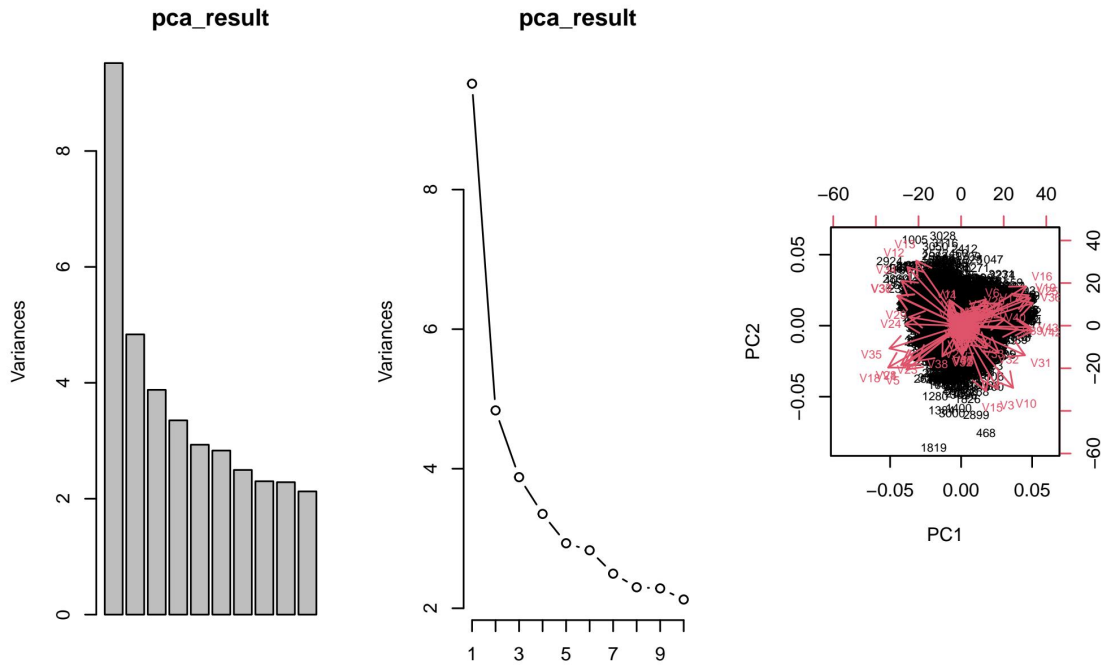
The second phase of research project involves implemented an unsupervised machine learning technique called Principal Component Analysis. Principal Component Analysis is essentially a dimensionality reduction technique, which is primarily used to reduce the dimensionality of large dataset. It does this by transforming a large set of variables, into a smaller set o variables while preserving as much information as possible. The new variables are called Principal Components, and these PCs are linear combinations of the original variables in the dataset. The linear combinations are constructed in a way that the new variables, which are the Principal Components, are uncorrelated with each other and most of information is compressed into the first few components, for instance PCA makes an attempt to put the maximum amount of information in the first component, and then the maximum remaining information in the second component.

PCA is often necessarily in large dataset in order to simplify the dataset and make it more interpretable. For instance in a dataset like ours that has 80 predictor variables it would difficult to asnwer questions like which variables have the greatest affect on the response variable. Therefore, we reduce the dimensionality of the dataset so that it is easier to interpret and visualzing any underlying distributions and patterns in the data.

Implementation of the Principal Component Analysis The first step in implementing our PCA, is to standardize our dataset. Standardization is a process that involves subtracting the mean and dividing by the standard deviation for each value of each variable. This is necessary, because variables with larger ranges(e.g 0 to 100) will dominate in comparison to variables with smaller ranges(e.g 0 to 1). Therefore to ensure that every variable contributes equally to our analysis we transform them to the same scale. The code below also computes our principal component analysis through the “prcomp” function and it center and scale arguments being set to true ensure that the variables are mean-centered and scaled.

```
training_df_x_scaled <- scale(train_x)
pca_result <- prcomp(training_df_x_scaled, center = TRUE, scale = FALSE)
```

The graphs bewlow depict how the variances change across the different principal componenets



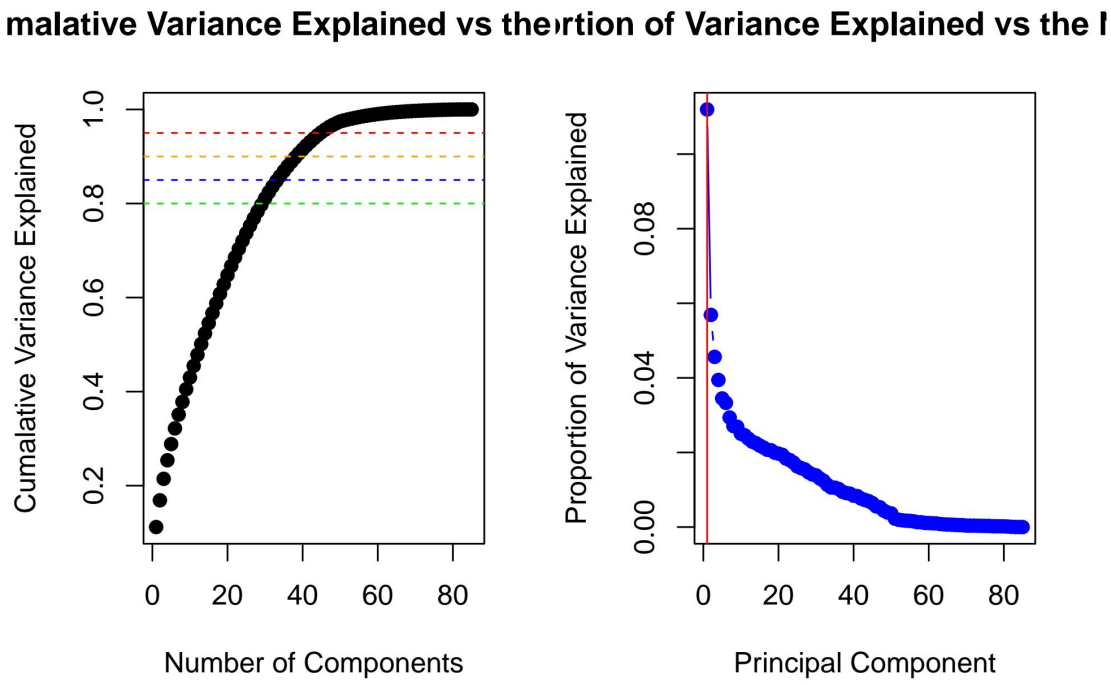
One of our core objectives of principal component analysis is to see how much information each principal component retains. We do this by calculating the variance of each principal component as it helps us put into perspective how much each component contributes to capturing the variability in the data, in other words the retaining the information in the original dataset. Another way in which we can measure how much principal component analysis is to see how much is by calculating the cumulative variance which is the adding up the variance explained by each component in a sequence, and this is helpful in determine the total variance explained. Both these methods help us understand the optimal number of principal components we need to reduce the dimensionailty of our dataset and fit into our classification models.

```
var_explained <- pca_result$sdev^2/ sum(pca_result$sdev^2)
explained_variance <- summary(pca_result)$importance[2,]
cumalative_variance <- cumsum(explained_variance)
```

To help us make the decision of what is the optimal number of principal components to choose to reduce the dimensionailty of the dataset and fit in our final classification models we have created a scatter plot that depicts the how the cumulative variance that is explained increase with each additional principal components. We have also created a scree plot that depicts the proportion of variance explained by each principal component, and this is often done by identifying an elbow point which is the point where the explained variance dramatically drops. This point will essentially be the cutoff for reducing the number of components included. However, since we have large number of principal components it is difficult to pinpoint what this cutoff point actually is, due to graph dimensions. Due to this we use the graph that depicts the cumulative

variance explained in order to pick a optimal number of principal components. When using the cumulative variance explained, the statistical recommendation for the number of principal components to retain suggest selecting a number of principal components correspond to the cumulative variance explained threshold of between 80% to 95% as these thresholds offer a balance for the tradoff for creating a simpler model with fewer components and retaining enough information from the original data set, in other words variance for further analytical purposes. We selected a threshold of 90% variance explained as it provides a balance between caputring a sufficent amount of variance and avoiding the potential overfitting that comes with too many dimensions. The reasons why we don't choose 80% or 85% is because they may be too low to explain the complexity present in the dataset, and a threshold of 95% might offer the possibility of too many compoenents.

Each of the thresholds are given dpeicited by the lines of different colors in the Cumalative Variance Explained graph



To verify the number of principal components we need we use the code below in order to identify the number of principal components that explain just under or exactly 90% of the variance and not more. It other words the code tells us the smallest number of components for which the cumulative variance first reaches or even exceeds the 90% threshold. We can see that we need to choose the first 39 principal components.

```
num_components_90 <- which(cumalative_variance >= 0.90)[1]
num_components_90
```

```
## PC39
##   39
```

In the code below we transform our training and testing datasets based of the principal compoenents analysis

```r
training_pca_x <- predict(pca_result, newdata = train_x)[,1:39]
testing_pca_x <- predict(pca_result, newdata = test_x)[,1:39]

training_pca <- as.data.frame(cbind(training_pca_x, train_y)) %>% rename(V86 = train_y)
testing_pca <- as.data.frame(cbind(testing_pca_x, test_y)) %>% rename(V86 = V1)

training_pca$V86 <- as.factor(training_pca$V86)
testing_pca$V86 <- as.factor(testing_pca$V86)
```

Implementing Classification Model 1: Logistic Regression

The first classification model that we will implement in our research project is logistic regression. Logisitc Regression is a statistical method used for binary classification purposes, this is when the response variable is binary, which means it only has two outcomes. Logistic Regression outputs a probability of the dependent variable belonging to a particular class. Logistic regression's use of the logistic function is its use of the logistic function, which also know as the sigmoid function, and it's use of maximum likelihood estimation to estimate parameter(the coefficients and intercepts) from the training data form the basis of logistic regression.

Our goal is this phase of the research project is to implement both a standard logistic regression model and a penalized logistic regression model(such as Lasso or Ridge) that is optimzied using cross-validation. The cross-validation will atttempt to find the optinal value for λ which is the penalty term applied to coefficients in regularzation techniques like ridge and lasso regression. We will then compare the error metrics of each of the models in order to find the most optimal logistic regression model for the final analysis.

```r
set.seed(7)
library(glmnet)
```

Normal Logistic Regression

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-8
```

```r
logistic_model <- glm(V86 ~ ., family = "binomial", data = training_pca)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

The code below creates predictions based of the logistic regression model using the predict function, where the setting of type = "response" makes R return the predicted probabilities of the positive class(the class that corresponds to the value of 1. We then use the ifelse() function to convert these predicted probabilities into binary class labels based on a decision boundary of 0.5. If the predicted probability is greater than 0.5,

then the functon will assign a class label of 1: otherwise it will assign a class label of 0. We choose 0.5 as our decision boundary, because this threshold is commonly used in binary classification tasks. We choose 0.5 as the threshold because it creates symettry in the decision process, as the odds of being in the positive class versus the negative class are exactly even, therefore it serves as a natural cutoff point where the probability of being in either class is equal.

Below is the confusion matirx of the logistic regression model

```
##            Actual
## Predictions    0    1
##           0 3741  233
##           1   21    5
```

The code below calculates the metrics of sensitivity, specificity, accuracy rate, error rate, precision and the f1 score. We implement this code for all of the models in the project, however have hidden the code when finally knit to a pdf

Penalized Logistic Regression Models

Ridge Penalized Logistic Regression In the code below we implement three types of Penalized Logsitic Regression - Ridge Regression, Lasso Regression, and Elastic Net Regression. Regression are essentially regularization techniques that add a penalty to the coefficients of the predictor variables of linear regression model and shrink them in order to prevent overfitting and improve generalization to new unseen data. Lasso Regression adds a penalty to the absolute value of the sum of the coefficients, and can perform variable selection by setting some of the coeffcients to zero. Ridge regressions adds a penalty equal to the square of the magnitude of the coefficients and while it shrinks the coefficients towards zero it does not shrink them to zero like Lasso regression. Elastic Net acts as intermediary between Lasso and Ridge Regression as it combines the properties of both, for instance it can add a penalty equal to the square of the magnitude of the coefficients and add a penalty to the absolute value of the sum of the coefficients. Therefore it can shrink some coeffcients towards zero and shrink others to zero.

In code below we attempt to simulate cross-validation for each of these methods in order to find the penalty term λ that minimizes the cross-validation error and then we create a final model for each regularization technique with that value of λ.

```
library(glmnet)
set.seed(7)

pen_logistic_ridge <- cv.glmnet(x = as.matrix(training_pca[, 1:39]),
                                y =  as.matrix(training_pca[, 40]),
                                nfold = 20,
                                family = "binomial", alpha = 0)
```

```
## The lambda value that minimizes the cross-validated error for penalized logistic ridge regression is
```

```
##            Actual
## Predictions    0    1
##           0 3750  233
##           1   12    5
```

Elastic Net Penalized Logistic Regression Now we implement cross-validation for Elastic Net Penalized Logistic Regression for the optimal penalty parameter.

```
set.seed(7)
library(glmnet)

pen_logistic_elastic_net <- cv.glmnet(x = as.matrix(training_pca[, 1:39]),
                          y =  as.matrix(training_pca[, 40]),
                          nfold = 20,
                          family = "binomial",
                          alpha = 0.7)
```

```
## The lambda value that minimizes the cross-validated error for penalized elastic net regression is 0.(
```

Below is the confusion matirx for the standaded logistic regression. The code below is used to calcualte the prediction of the logistic regression model and then create a confusion matirx for model based on the predictions and the actual values. This code is used for every other model in the report, however we hide the code in order to reduce the number of pages.

```
set.seed(7)
logistic_predictions_elastic_net <- predict(pen_logistic_elastic_net, newx = as.matrix(testing_pca[, 1:3
logistic_predictions_elastic_net_labels <- ifelse(logistic_predictions_elastic_net > 0.5, 1,0)
confusion_matrix_logistic_elastic_net <- table(Predictions = logistic_predictions_elastic_net_labels, A
confusion_matrix_logistic_elastic_net
```

```
##            Actual
## Predictions    0    1
##           0 3754  234
##           1    8    4
```

```
library(glmnet)

pen_logistic_lasso <- cv.glmnet(x = as.matrix(training_pca[, 1:39]), y =  as.matrix(training_pca[, 40])
```

Lasso Penalized Logistic Regression

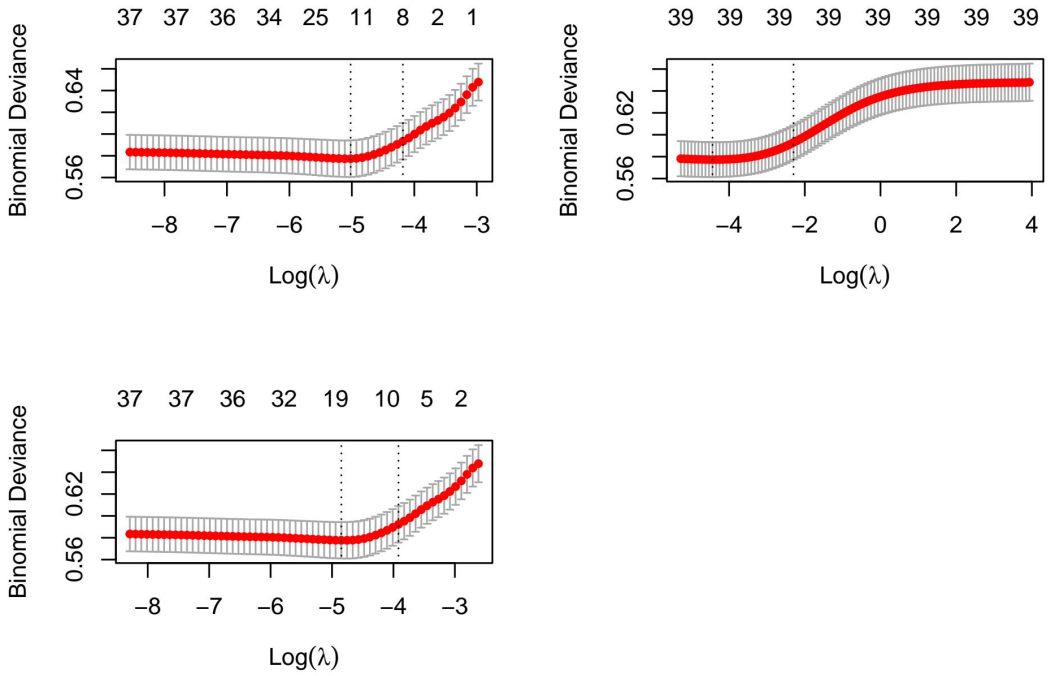
```
## The lambda value that minimizes the cross-validated error for penalized lasso regression is 0.006607
```

Below is the confusion matirx for the lasso logistic regression

```
##            Actual
## Predictions    0    1
##           0 3754  234
##           1    8    4
```

The graphs for each method help visualize how the error changes with different levels of regularization and can help us identify the value of *lambda* that minimizes the cross-validation error. The graphs below are in order of lasso, ridge and elastic_net

```
par(mfrow = c(2, 2))
plot(pen_logistic_lasso)
plot(pen_logistic_ridge)
plot(pen_logistic_elastic_net)
```



The table below depicts the error metrics of each of the variations of logistic regression

```
##                            Model Sensitivity Specificity Accuracy_Rate
## 1            Logistic Regression  0.02100840   0.9944179       0.93650
## 2      Ridge Logistic Regression  0.02100840   0.9968102       0.93875
## 3 Elastic Net Logistic Regression  0.01680672   0.9978735       0.93950
## 4      Lasso Logistic Regression  0.01680672   0.9978735       0.93950
##   Error_Rate Precision   F1_Score
## 1    0.06350 0.1923077 0.03787879
## 2    0.06125 0.2941176 0.03921569
## 3    0.06050 0.3333333 0.03200000
## 4    0.06050 0.3333333 0.03200000
```

Analysis of the Logisic Regression Error Metrics The above data frame lists the results of implementing logistic regression, ridge logistic regression, elastic net logistic regression, and lasso logistic regression using the lambda value that minimizes the cross-validated error for each individual model. Firstly, we see that the logistic regression model with the lambda value that minimizes the cross-validated error has a sensitivity of 0.02100840, a specificity of 0.9944179, an accuracy rate of 0.93650, an error rate of 0.06350, a precision of 0.1923077, and a F1 score of 0.03787879. Next, we see that the ridge logistic regression model with the lambda value that minimizes the cross-validated error has a sensitivity of 0.02100840, a specificity

of 0.9968102, an accuracy rate of 0.93875, an error rate of 0.06125, a precision of 0.2941176, and a F1 score of 0.03921569. Furthermore, we see that the elastic net logistic regression model with the lambda value that minimizes the cross-validated error has a sensitivity of 0.01680672, a specificity of 0.9978735, an accuracy rate of 0.93950, an error rate of 0.06050, a precision of 0.3333333, and a F1 score of 0.03200000. Lastly, we see that the lasso logistic regression model with the lambda value that minimizes the cross-validated error has a sensitivity of 0.01680672, a specificity of 0.9978735, an accuracy rate of 0.93950, an error rate of 0.06050, a precision of 0.3333333, and a F1 score of 0.03200000. When evaluating and choosing between different models such as logistic regression, ridge logistic regression, elastic net logistic regression, and lasso logistic regression, particularly for binary classification tasks like determining if a customer will purchase caravan insurance, the choice of the metric to evaluate which model is the best classifier can significantly influence the decision on the "best" model. In this case, given that the dataset is imbalanced (few people buy insurance compared to those who do not), F1 Score emerges as a good general metric for model comparison in binary classification situations, especially when the dataset is imbalanced. This is so since it ensures a balance between the need to correctly identify all positive instances and the need to maintain a high quality of the positive predictions. A higher F1 Score suggests better model performance. Aiming for a higher F1 score means to aim for a model that neither misses too many true cases nor wastes resources on too many false alarms (high precision). In situations like customer targeting (caravan insurance policy), where both types of errors can be costly, a high F1 score is very desirable. From prior analysis, we can see that ridge logistic regression has the highest F1 score with a score of 0.03921569. However we also want to pick a model that minimizes error and increases accuracy therefore in for our final analysis we pick lasso logistic regression. Lasso Logistic regression would minimize error and maximize error since due to their additional feature selection capabilities, which can enhance model interpretability and potentially improve generalization on unseen data.

We also calucate the AUC of the ROC Curve

```
auc(roc_curve_log)
```

```
## Area under the curve: 0.5073
```

Implementing Classification Model 2: KNN Classification

The second classification model that we will be implementing in our research project is a KNN classifier. KNN stands for k-nearest neigbours, and it is non-parametric, supervised machine learning algorithm that can bs used for either regression or classification tasks. KNN uses proximity and closeness as it's main measure for classification and prediction. It works by finding the closest points to the datapoint the needs to predicted or classified. A distance metric such as Euclidean Distance, Manhattan distance, Minkowski Distance, etc is used to measure closeness. The hyper parameter k, determines the number of nearest neighbors that we include in the process based on the choose distance metrics. The kNN algorithim then predicts the mean or median of the values of the neighbors for regression taks or uses majority voting for classification tasks, i.e finds class that appears the most frequently among the neighbors and then assigns the data point to that class.

Setting Up the Cross-Validation for KNN In the code below we implement cross-validation to find the most optimal kNN model. We do this by setting up a grid of possible k values. The algorithm then finds the accuracy measure for kNN classifier with each of the k values and select the k value that maximizes the accuracy - this k value will give our most optimal model that will generalize well to new unseen data.

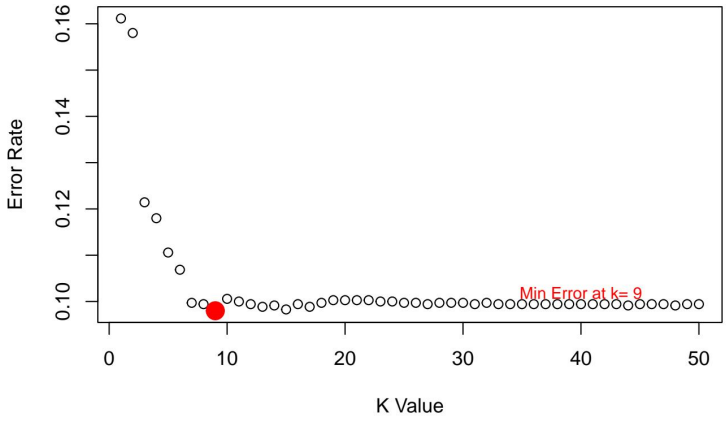
```
set.seed(7)
library("caret")
control <- trainControl(method = "cv", number = 10)

knn.cvfit <- train(as.factor(V86) ~ ., method = "knn", data = training_pca, tuneGrid = data.frame(k = s
```

```
cat("The K-value that gives the minimum error rate and maximum accuary is", which.max(knn.cvfit$results$
```

```
## The K-value that gives the minimum error rate and maximum accuary is 9
```

The graph below shows how k changes with our error rate and therefore how we choose the k that minimized the error rate and maximzied accuarcy in order to fit our final KNN model. We can see the minimum error rate and maximum accuarcy rate occurs at k = 9



Predictions and Calulcations of Error Metrics for the KNN Classification Model

```
##            Actual
## Predictions    0    1
##           0 3736  232
##           1   26    6
```

```
## The Sensitivity for the KNN Classifier Model is 0.02521008
```

```
## The Specificity for the KNN Classifier Model is 0.9930888
```

```
## The Accuaracy Rate for the KNN Classifier Model is 0.9355
```

```
## The Error Rate for the KNN Classifier Model is 0.0645
```

```
## The Precision for the KNN Classifier Model is 0.1875
```

```
## The F1-Score for the KNN Classifier Model is 0.04444444
```

We also depict the ROC curve and the AUC for the KNN mode. This is important as the ROC curve plots the TPR and FPR and helps provide threshold for binary classification. The AUC measures the entire area underneath the ROC curve and it essentially provides an aggreagte measure of the performance of all possible classifcation thresholds.

```r
auc(roc_curve_KNN)
```

```
## Area under the curve: 0.5091
```

The AUC value of 0.5041 for our models ROC curve indicates that the model has a limited ability to distiguish between the positive and negative classes.

Analysis fo KNN Classifier's Error Metrics KNN: For the better understanding of the reader, to explain the evaluation metrics that were calculated and reported for the optimal KNN classifier model (with k being equal to 50) that was determined by the largest accuracy value, we explain what each evaluation metric means. Measures how well the KNN classifier identifies actual buyers of caravan insurance. High sensitivity means the model successfully captures a large proportion of the people who buy insurance (true positives). Specificity (True Negative Rate) measures the model's ability to correctly identify those who do not buy caravan insurance. High specificity indicates that the model is effective at filtering out non-buyers (true negatives). Accuracy Rate measures the overall correctness of the model. It measures how often the model correctly predicts whether a customer buys insurance or not. Error rate is the complement of accuracy, indicating the proportion of all predictions that were incorrect. Lower error rate means fewer mistakes made by the model. Precision (Positive Predictive Value) measures the accuracy of positive predictions made by the model. High precision means that when the model predicts a customer will buy insurance, it's likely correct. F1-Score combines precision and sensitivity into a single measure that seeks a balance between them, using their harmonic mean. It is particularly useful when the classes are imbalanced. For the optimal KNN classification model that was found, the sensitivity is 0.02521008, the specificity is 0.9930888, the accuracy rate is 0.9355, the error rate is 0.0645, the precision is 0.1875, and the F1-Score is 0.04444444.

Implementing Classification Model 3: Decision Tree

The third classification model that we will be fitting in our dataset is a decision tree. Decision Trees are a type of non-parametric and supervised learning algorithim, that can be used for both classification and regression tasks. They essentially have a hierarchical, tree strucutre and consists of multiple components called the root node, braches, internal node and leaf nodes.

Setting Up the Cross-Validation for the Decision Tree For the cross-validation of the decision tree we use ROC to select the optimal model using the largest value and then try to find the most optimal value of cp. Cp is complexity balances parameter of the decisions tree and essentially the complexity of the tree, which is determined by the number of splots and xerror(which is the predictive accuarcy of he model). The final value used for the model was cp = 0.005387931.

```
## CART
##
## 3500 samples
##   39 predictor
##    2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3150, 3150, 3149, 3150, 3149, 3150, ...
## Resampling results across tuning parameters:
##
##   cp           ROC        Sens       Spec
##   0.005387931  0.6330463  0.9723910  0.097983193
##   0.005747126  0.6265792  0.9723910  0.095126050
```

```
##   0.010057471  0.5226230  0.9955616  0.008571429
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.005387931.
```

```
library(caret)
library(rpart.plot)
```

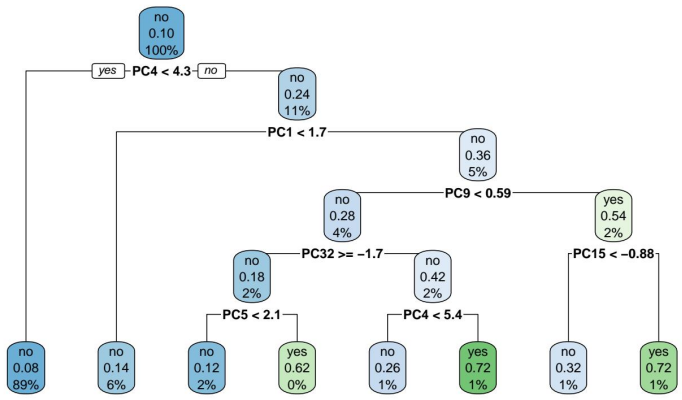
```
## Loading required package: rpart
```

```
best_model <- model$finalModel
```

```
best_cp <- model$bestTune$cp
cat("The optimal cp for our model is", best_cp)
```

```
## The optimal cp for our model is 0.005387931
```

Pruned Decision Tree for Caravan Insurance Prediction




```
##            Actual
## Predictions    0    1
##           0 3723  228
##           1   39   10
```

```
## The Sensitivity for the Decision Classifier Model is 0.04201681
```

```
## The Specificity for the Decision Classifier Model is 0.9896332
```

```
## The Accuaracy Rate for the Decision Classifier Model is 0.93225
```

```
## The Error Rate for the Decision Classifier Model is 0.0645
```

```
## The Precision for the Decision Classifier Model is 0.2040816
```

```
## The F1-Score for the Decision Classifier Model is 0.06968641
```

```
cat("The area under the cure for the Decision Tree Classification Model is ", auc(roc_curve_decision_tr
```

```
## The area under the cure for the Decision Tree Classification Model is  0.515825
```

The AUC value of 0.515825 for our models ROC curve indicates that the model has a limited ability to distiguish between the positive and negative classes.

Analysis of Decision Tree Error Metric Tree: The sensitivity of 0.0420 indicates a limited ability to accurately identify positive cases (buyers of caravan insurance), potentially resulting in a significant number of missed true positive instances. While the specificity of 0.9896 suggests strong performance in identifying negative cases (non-buyers), this may be relatively easier to achieve due to the dataset's class imbalance. The accuracy rate of 0.9322 reflects a correct prediction rate of approximately 93.22%, though this figure may be inflated due to the dataset's skewed class distribution. A precision of 0.2041 implies that only about 20.41% of the instances predicted as positive are truly positive, indicating a notable number of false positives. Similarly, the low F1-Score of 0.0697 underscores the model's challenge in balancing precision and recall, indicative of subpar overall performance.

Implementing Classification Model 4: Support Vector Machines(SVM)

Setting Up the Cross-Validation for SVM We implement a 10-fold cross-validation for Support Vector Machines in order to assess the generalization of the model and also pick the best model. We do this by identifying the righ choice for the hyperparamters of th SVM model, which are C and γ. Both these hyperparamters help us decipher the trade-off between bias and variance in our models and through this we can find the most optimal SVM model. C is the cost function, which creates a trade-off between the correct classification of training examples and the process of maximization of the decision boundary - it tends to be extremely important in cases when the data is not linearly separable. For instance, a large C value makes an attempt to classify all of the training examples correctly, in other words minimize missclassfications, however this can create a narrower and smaller margin. Due to this the numer of support vectors also decreases, as fewer points are needed to define a narrow margin. Essentially with a higher C the penalty ofr missclassificatons increases which means the SVM algorithim makes harder in order to avoid missclassfications. γ paramter is the kernel coefficient for a radial basis function(RBF) kernel, we use this kernel as it maps samples into a higher-dimensional space and this allows it works well with the complexity of our dataset and with non-linearity as it have the ability to create non-linear decision boundaries in combination with its realtivlry straightforward computations.

γ defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. Thes eare important because they can essentially configure how the far the influence of a single training examples reaches. For instances a lower value means that the kernel has a larger radius of influence. Due to this the training example can have a far reach and heavily influence the classfication of other points. This will also make the decision boundry smoother and help avoid overfitting as it does not retain nose from the training data.

```
set.seed(7)
# Load required libraries
library(caret)
library(e1071)
```

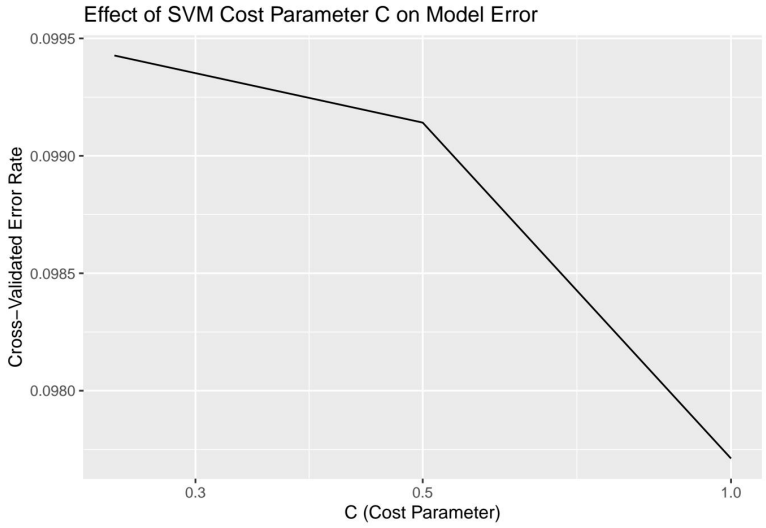
```r
train_control <- trainControl(method = "cv", number = 10, savePredictions = "final")
svm_model_cv <- train(x = training_pca[,1:39], y = training_pca$V86, method = "svmRadial", trControl =

svm_model_cv
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 3500 samples
##   39 predictor
##    2 classes: 'no', 'yes'
##
## Pre-processing: scaled (39)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3150, 3150, 3149, 3150, 3149, 3150, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.9005727  0.000000000
##   0.50  0.9008585  0.005027933
##   1.00  0.9022887  0.034308641
##
## Tuning parameter 'sigma' was held constant at a value of 0.02308245
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.02308245 and C = 1.
```

From the model output of the cross-validation of SVM we can see that the tuning parameter 'sigma' was held constant at a value of 0.02308245 and accuracy was used to select the optimal model using the largest value. The final values used for the model were sigma = 0.02308245 and C = 1 - these are the optimal values for our SVM model.

The graph below dpecits how the cost paramter changes with model error and how this helped us in our cross-validation to pick the most optimal value of C. The most optimal value of see is selected based on what value minmizes the cross-validated error.



```r
set.seed(7)
final_svm_model <- svm(x = training_pca[,1:39],
                       y = training_pca$V86,
                       method = "radial basis",
                       C = 1, gamma = 1/(0.02308245*39),
                       type = "C-classification")

final_svm_model
```

```
##
## Call:
## svm.default(x = training_pca[, 1:39], y = training_pca$V86, type = "C-classification",
##     gamma = 1/(0.02308245 * 39), method = "radial basis", C = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  3228
```

```
##            Actual
## Predictions    0    1
##         no  3731  228
##         yes   31   10
```

```
## The Sensitivity for the SVM Classifier Model is 0.04201681
```

```
## The Specificity for the SVM Classifier Model is 0.9917597
```

```
## The Accuaracy Rate for the SVM Classifier Model is 0.93525
```

```
## The Error Rate for the SVM Classifier Model is 0.06475
```

```
## The Precision for the SVM Classifier Model is 0.2439024
```

```
## The F1-Score for the SVM Classifier Model is 0.04444444
```

The metrics presented for the SVM Classifier Model indicate a pattern similar to that of the KNN Classifier Model, suggesting suboptimal performance across various evaluation criteria. With a sensitivity of 0.0420, the model demonstrates limited ability to correctly identify positive cases (buyers of caravan insurance), potentially missing a significant portion of true positive instances. Although the specificity of 0.9920 suggests strong performance in identifying negative cases (non-buyers), this might be easier to achieve given the dataset's imbalance. The accuracy rate of 0.9355 indicates that the model is correct in roughly 93.55% of its predictions overall, yet this might be inflated due to the dataset's class distribution. A precision of 0.25 implies that only 25% of the instances predicted as positive are truly positive, highlighting a high rate of false positives. Similarly, the low F1-Score of 0.0444 emphasizes the model's struggle to balance precision and recall, indicating poor overall performance.

```r
auc(roc_curve)
```

```
## Area under the curve: 0.5169
```

The AUC value of 0.517 for our models ROC curve indicates that the model has a limited ability to distiguish between the positive and negative classes.

Analysis of the Error Metrics of SVM We find the optimal support vector machine model to have a cost parameter of 1 and a gamma parameter of 1 / (0.01933608*39). For this model, the sensitivity is 0.02941176, the

specificity is 0.997076, the accuracy rate is 0.9395, the error rate is 0.0605, the precision is 0.3888889, and the F1-Score is invalid.

Comparing the Models

```
##                       Model Sensitivity Specificity Accuracy_Rate Error_Rate
## 1 Lasso Logistic Regression  0.01680672   0.9978735       0.93950    0.06050
## 2                       KNN  0.02521008   0.9930888       0.93550    0.06450
## 3             Decision Tree  0.04201681   0.9896332       0.93225    0.06450
## 4                       SVM  0.04201681   0.9917597       0.93525    0.06475
##   Precision   F1_Score
## 1 0.3333333 0.03200000
## 2 0.1875000 0.04444444
## 3 0.2040816 0.06968641
## 4 0.2439024 0.07168459
```

The table depicts the error metrics for all of the optimal cross-validate models we have fit in our research. If we were to select a model based solely on the core objective of predictive accuracy then we would tying to maximize the accuarcy rate and minimize the error rate - due to this we would pick the model with the lowest error rate which is the decision tree and the model the maximum accuracy rate which is the logistic lasso regression. However, when determining whe best model is predcitive accuary is not the only metric that should be considered. For instance, in the context of our dataset where were trying to classify potential buyers of caravan insurance policies, then selecting the best model could depend on the specific objectives of the project. For instance, if the objective was to ensure that as many potential buyers as possiblle are identified(in other words minimize false negatives), then it is pivotal that the possbility of missing potential buyers is minimized, then we would pick the model with highest sensitivity, in this case it would be the decision tree or an svm model. Another possible goal of this dataset could be trying to reduce the number of non-buyers of the policy, then we would want to to minimize the number of false positives and maximize specificly - due to this would pick the model with the highest specificy which would be Lasso Logistic Regression. We could also have a goal of finding the optimal balance between identifying true buyers and non overwheliming non-buters, with something like the markeing of the insurance policy, therefore the model with highest F1-score, such as Decision Tree or an SVM model would be the most suitable. Moreover, some could also say that SVM's may be the best model has it has the ability to configure its kernel into linear and polynomial and therefore fit to varying degrees of complexity especially when the model could be useful for datasets that are linearly inseparable. Lasltly we could also say due to the largely imbalanced nature of our dataset, finding a high precision might be neccesary and therefore we may choose a KNN as the best model.

Final Conclusion The basis of our project was fit different types of classification models that have been optimized through cross-validation to find the hypermarter configurations for each model that result in the most optimal model and then compare then using a variety of error metrics to see whic model is the best.

While we did succesfully do that there, were a number of disadvantages to our research methodology and possibile improvements that we could implement. After executing our four models and analyzing the results, we identified potential improvements that could enhance the performance of each classification model. For our Logistic Regression model, which utilized PCA-transformed variables, adjusting the principal components to capture a more significant amount of variance could be beneficial.

Additionally, we used 20-fold cross-validation, which could lead to increased computational costs and potentially diminishing returns in model stability. This approach might also introduce bias if not managed properly. Utilizing automated hyperparameter optimization techniques, such as Bayesian optimization, might lead to better and more efficient model tuning compared to standard grid search. Methods that we could consider including next time to get more optimal results is cross-validation strategy which reduces the number of folds.

A way in which we could improve our Logistic Regression is that we use could also use cross-validation on different solver algorithms (like liblinear, saga, lbfgs). We could have also applied a wider change of penalty values for each model to see how our results changed.

Regarding our KNN model, we varied the number of neighbors (k) from 1 to 50, selecting the "k" with the lowest error rate. When adjusting our hyperparameters, it is important to select the most appropriate "k" and consider the number of neighbors to consider as a smaller "k" makes the boundary between classes less distinct, and a larger "k" can smooth the decision boundary too much. It is also crucial to ensure that all features are scaled similarly, as KNN is sensitive to the scale of features since it relies on distances. A method that could enhance our results would be feature scaling ensuring all features are scaled. We coud also try to use different distance metrics such changing between Euclidean and Manhattan distance or even change the weights applied to confiugre the model.

For the Decision Tree model, one challenge is the risk of overfitting, which often leads to unnecessary splits and biases in decision trees. Although we chose a reasonable complexity parameter (cp), exploring a wider range of values or other parameters could improve our pre-pruning results and reduce overfitting. We could adjust our hyperparameters by considering controlling the maximum depth of the tree which could help prevent our model from overfitting and increasing the minimum number of observations ensuring that splits contribute significantly to model accuracy.

Finally, our SVM model, which uses a Radial Basis Function (RBF) kernel suitable for non-linear problems, should be reevaluated to confirm that a non-linear model is necessary compared to a linear kernel. In hyper parameter tuning, using tools like "expand.grid" to define a broader range of valuesssld be more effective than relying on default values. Hyper parameters that we can tune in our SVM model is by testing a range of values for regularization parameter and adjusting it to help reduce overfitting or underfitting. To further improve our model we could also try to configure the kernel type we fit, for instance we fit a polymomial and linear kernel which dtermine the type of hyperplane used to separate our data to see how our results change.

Some other issues we faced in our research methodology was the presence of a highly balanced dataset. This was depicted in The AUC values of around 0.5 for our models ROC curve which indicated that the model has a limited ability to distiguish between the positive and negative classes. and while we did under sample the majority class, we did not do it o its full extent as we wanted to retain as much informaton as possible.Therefore, in the future we could implement techniques such as SMOTE for oversampling, possibily adjusting class weights, or using anomaly detection could potentially enhance sensitivity without sacrificing too much specificity. We could also use highly advancd ensemble techniques such as balanced random forest or XGBboost in order to make the models more robust to class imbalance.

References

A Step-By-Step Complete Guide to Principal Component Analysis | PCA for Beginners." TURING, www.turing.com/kb/guide-to-principal-component-analysis. Accessed 30 Apr. 2023.

"What Is Logistic Regression? Equation, Assumptions, Types, and Best Practices." Spiceworks, 18 Apr. 2022, www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/#:~:text=Practices%20for%2(,What%20Is%20Logistic%20Regression%3F,1%2C%20or%20true%2Ffalse. Accessed 30 Apr. 2023.

"Ridge Regression Explained, Step by Step." Machinelearningcompass, machinelearningcompass.com/machine_learning_moc Accessed 30 Apr. 2023.

"Lasso Regression." Bookdown, bookdown.org/tpinto_home/Regularisation/lasso-regression.html. Accessed 30 Apr. 2023.

"K-Nearest Neighbor(KNN) Algorithm." GeeksforGeeks, www.geeksforgeeks.org/k-nearest-neighbours/. Accessed 30 Apr. 2023.

"Support Vector Machine — Introduction to Machine Learning Algorithms." Towards Data Science, towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. Accessed 30 Apr. 2023.

"Decision Tree Classification Algorithm." JavaTpoint, www.javatpoint.com/machine-learning-decision-tree-classification-algorithm. Accessed 30 Apr. 2023.