Performing Machine Learning Algorithms to Diagnose Diabetes in Pima Native American Women

Idris Rasheed

August 12, 2017

1. Abstract

This paper intends to find a machine learning algorithm to accurately diagnose diabetes in Pima Native American women to better understand how machine learning can be used to diagnosis other diseases or correctly classify medical data. Three machine learning algorithms were implemented and tested for accuracy: Linear Support Vector Machine (LSVM), Radial Support Vector Machine (RSVM), and Random Forest. I conclude these models are viable options for accurately classifying medical data.

2. Introduction

This analytical paper discusses machine learning algorithms as tools to address the diabetic problem within the Pima Native American female population. Reports not that the Pima Native American women of Arizona have the highest reported prevalence of diabetes of any population in the world, exclusively type 2 diabetes.[1] We want to know how different machine learning algorithms are useful tools to the medical community such as bioinformaticians, clinicians, epidemiologists, and other professionals in the medical field. To do this, we construct LSVM, RSVM, and Random Forest models to better understand diabetes in the Pima Native American and possibly apply these tools to other fields of medicine. In conclusion, our accuracy tests indicate LSVM, RSVM, and Random Forest models, with some tuning, are useful computational methods in the medical field.

2.1 Description of the Dataset

The dataset analyzed is the "Pima Indian Women Diabetes" from the University of California, Irvine (UCI) data repository.[2] It contains 768 observations and 652 missing values for nine features that take integer and real values, which is ideal for this classification modeling project. These are the features within the dataset:

Feature Name	Amended Feature Name	Feature Type
Number of times pregnant	TotalPregnancies	Predictor
Plasma glucose concentration at 2 hours in an oral glucose tolerance test	PlasmaGlucose	Predictor

Diastolic blood pressure (mm Hg)	DiastolicPressure	Predictor
Triceps skin fold thickness (mm)	FoldThickness	Predictor
2-Hour serum insulin (mu U/ml)	Insulin	Predictor
Body mass index (weight in kg/(height in m)^2)	BMI	Predictor
Diabetes pedigree function	PedigreeFunction	Predictor
Age (years)	Age	Predictor
Diabetic or Non-Diabetic coded as 0 and 1	Diagnosis	Response

2.2 Software Implementation

For this paper, we will be using the statistical programming language R through the RStudio software to write and process our code. All our statistical analysis will be processed with these RStudio's libraries: corrplot, e1071, RSNNS, randomForest, mice, kernlab, caret, AppliedPredictiveModeling, and VIM, and our report will be manifested with RMarkdown.

3. Data Preprocessing

Our initial phase of this paper begins with installing and unpacking our packages with library() and importing our dataset.

3.1 Reading Data

```
#Create a vector of the feature names
headers <- c("TotalPregnancies", "PlasmaGlucose", "DiastolicPressure",
"FoldThickness",
"Insulin", "BMI", "PedigreeFunction", "Age", "Diagnosis")

#Import data
url <- paste0("http://archive.ics.uci.edu/ml/machine-learning-databases/",
"pima-indians-diabetes/pima-indians-diabetes.data")
diabetes <- read.csv(url(url),header = FALSE, col.names = headers)
```

The str() shows that all the features were imported correctly.

```
str(diabetes)
```

```
## 'data.frame':    768 obs. of  9 variables:
##  $ TotalPregnancies : int  6 1 8 1 0 5 3 10 2 8 ...
##  $ PlasmaGlucose    : int  148 85 183 89 137 116 78 115 197 125 ...
##  $ DiastolicPressure: int  72 66 64 66 40 74 50 0 70 96 ...
##  $ FoldThickness    : int  35 29 0 23 35 0 32 0 45 0 ...
##  $ Insulin          : int  0 0 0 94 168 0 88 0 543 0 ...
##  $ BMI              : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0
...
##  $ PedigreeFunction : num  0.627 0.351 0.672 0.167 2.288 ...
```

```
##  $ Age                : int  50 31 32 21 33 30 26 29 53 54 ...
##  $ Diagnosis          : int  1 0 1 0 1 0 1 0 1 1 ...
```

3.2 Coding Response Feature

For easier analysis, we will encode the 0 and 1 value for the diagnosis to "NotDiabetic" and "Diabetic", respectively, by converting the numerical response feature to a factor.

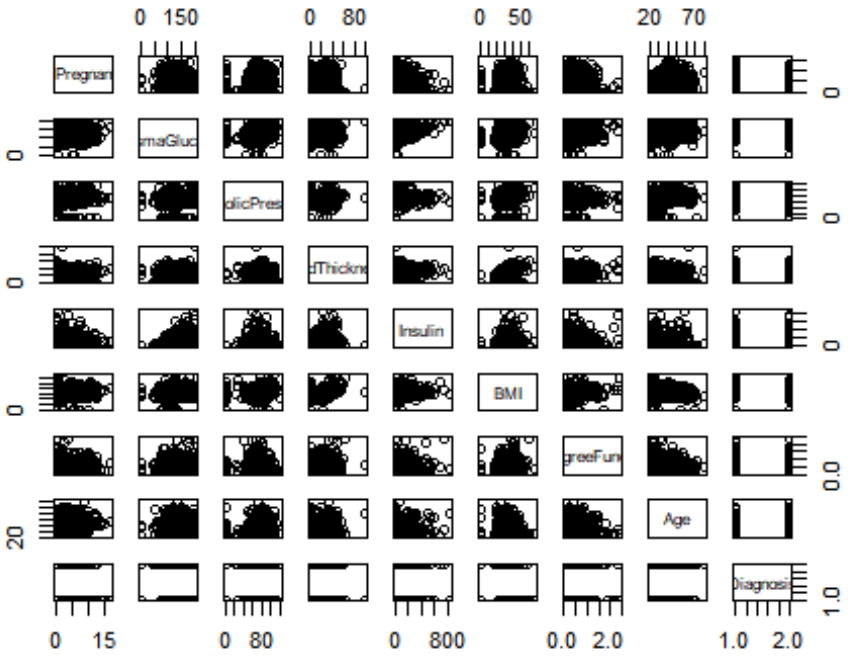
```
diabetes$Diagnosis <- as.factor(ifelse(diabetes$Diagnosis == 0,
"NotDiabetic", "Diabetic"))
```

3.3 Exploratory Analysis

We will the begin the exploratory analysis portion by developing a scatterplot matrix for all the features.

3.4 Scatterplot

```
pairs(diabetes)
```



Some features have a value of 0 which is biologically impossible for the PlasmaGlucose, DiastolicPressure, FoldThickness, Insulin, and BMI features. This problem needs to be corrected before we can continue.

3.5 Missing Datapoints

We discovered that the PlasmaGlucose, DiastolicPressure, FoldThickness, Insulin, and BMI features contain values of 0. To remedy this issue, we will assume all 0 values for these features are missing values and be recoded as NAs. This makes it easier to assess our missing values.

```
#Creates loops to make all 0's NA
for (i in 2:6){
  for (n in 1:nrow(diabetes)){
    if (diabetes[n, i] == 0){
      diabetes[n, i] <- NA
    }
  }
}
```

4.3 Missing Values

```
table(is.na(diabetes))
```

```
##
## FALSE  TRUE
##  6260   652
```

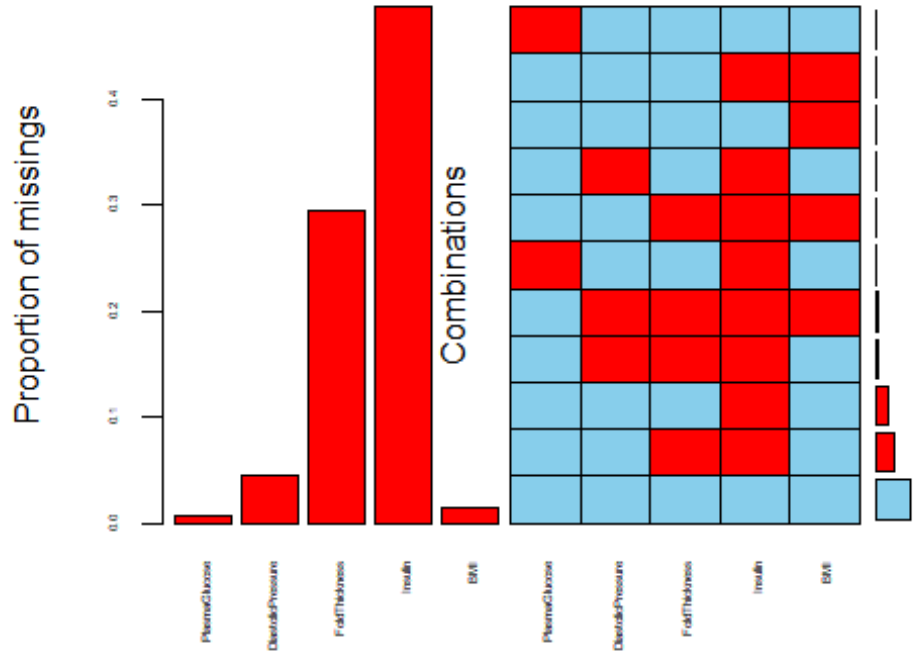
The Pima Native American women diabetes dataset contains 652 missing values, and we are interested in knowing the proportion of missing values by feature.

3.6 Proportion of Missing Values By Feature

We use aggr() to visual our missing data by feature to understand our dataset better.

```
aggr(diabetes[,2:6], cex.lab=1, cex.axis = .4, numbers = T, gap = 0)
```

```
## Warning in plot.aggr(res, ...): not enough horizontal space to display
## frequencies
```

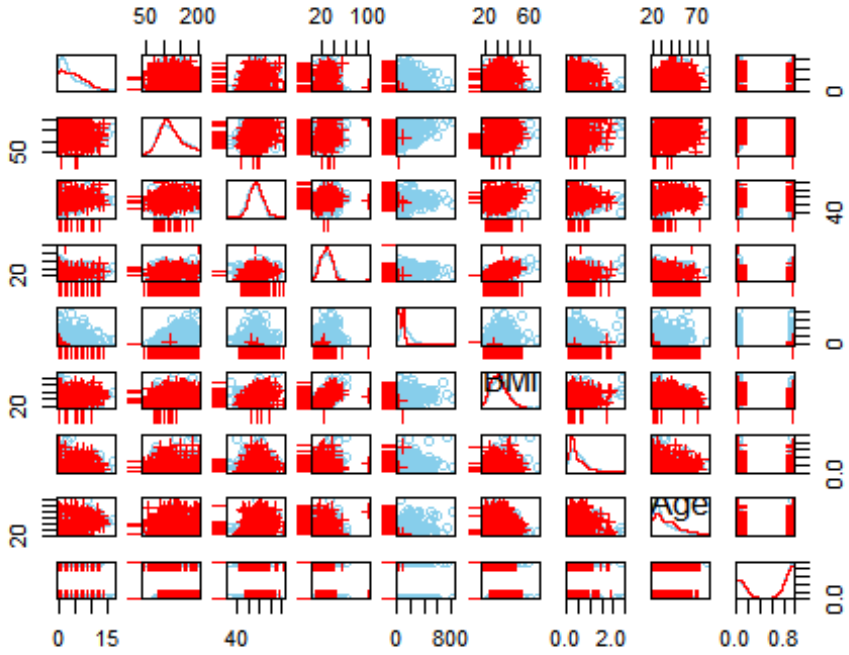


Above is the aggregation plot where red indicates the missing values and the light blue indicates the complete data. The "Proportion of missings" histogram shows that the Insulin feature has more than 50% values missing and the FoldThickness feature is missing around 33% of its values. Then we have the "Combinations" plot indicating that 51.04% of the data is complete and the other 48.96% of the values is missing from the dataset.

In this case, we want to keep all the missing values instead of dropping them through the process of imputation. Before deciding on how to impute the missing values, we want to understand how the missing values are distributed by constructing a scatterplot matrix of the missing values. We do this to identify any discernible relationship that impacts which imputation method to implement.

3.7 Distribution of Missing Values

```
#visualises a scatterplot of missing values
scattmatrixMiss(diabetes)
```



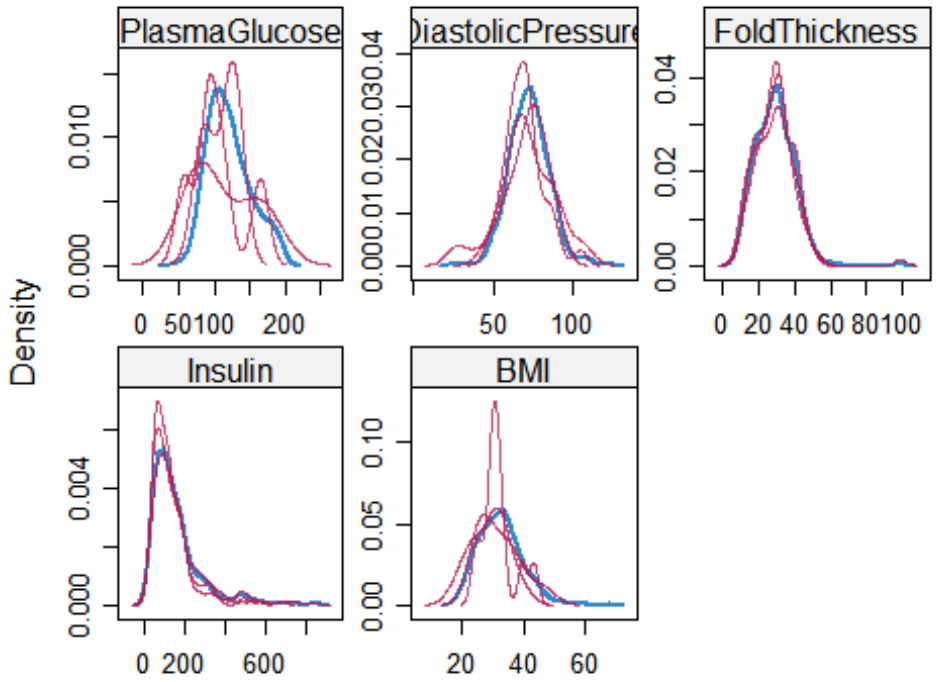
The missing scatterplot matrix indicates there is no relationship between the features and missing values. We can assume that the data is missing at random which means we can perform imputation by Predictive Mean Matching (PMM).

3.8 Predictive Mean Matching

PMM is a semi-parametric imputation method that predicts a value from a model's missing and observed values. Each missing value is replaced by an observed value close to the regression-predicted value or a random selection of observed values, corresponding to a feature.[3] We will now create a temporary, imputed diabetes dataset with the mice(), where the parameter "m" is the number of multiple imputations. We want to impute this data three times to represent the number of classification methods to be constructed.

Then we plot the density of the imputed diabetes dataset against the original diabetes dataset to see if they are approximately equal.

```
#Density plot original vs imputed dataset
densityplot(com.diabetes)
```



The red distributions are the imputed diabetes data, and the blue is the original diabetes data. The density plots show both the imputed and original distributions are approximately equal to each, so we can now replace our original dataset with the imputed data through the complete().

```
diabetes <- complete(com.diabetes)
```

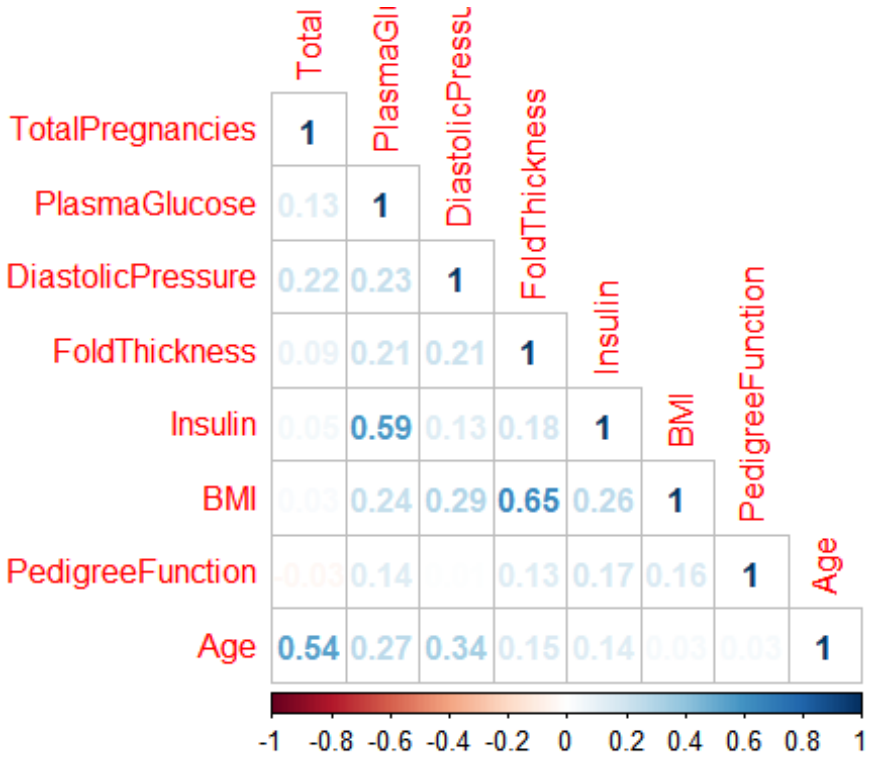
We now proceed with the feature selection process.

3.9 Feature Selection

In the feature selection process, we need to ensure no more than one feature has a correlation greater than .7 by visualizing a correlogram.

3.10 Correlogram

```
corrplot(cor(diabetes[,-9]),type = "lower", method = "number")
```



The examination of the correlogram shows that none of the features have a correlation coefficient of .7 or higher, so we include every feature.

3.11 Normalizing Data

All our features are different units so we need to normalize them before setting up our machine algorithms. We normalize our data by making each feature have a mean of 0 and a standard deviation of 1, with the scale() in R. Now we can proceed with setting up our algorithms.

```
diabetes[, 1:8] <- scale(diabetes[, 1:8], center = TRUE, scale = TRUE)
```

4. Setting Up the Algorithms

We can commence with constructing our three models from a training set of observations through 10-fold cross-validation with the trainControl(). 10-fold cross-validation splits up the dataset into nine training sets and one test set, so the original dataset has an equal chance of appearing in both sets, thereby preventing overfitting. [4]

4.1 Cross-Validation

```
Folds <- trainControl(method = "repeatedcv",
                      number = 10,
                      repeats = 10,
```

```
                        classProbs=TRUE,
                        summaryFunction=twoClassSummary)
```

4.2 Training and Test Set

The training set contains the set of examples fitted to the parameters of our classification methods, and the test set contains the set of examples to assess the accuracy of our classification methods.[5]

```
prop.table(table(diabetes$Diagnosis))

##
##    Diabetic NotDiabetic
##   0.3489583   0.6510417
```

Because approximately 65% of the Pima Native American women do not have diabetes and around 35% do have diabetes, we choose to separate the diabetes dataset into 70% training set and 30% test set.

65% also becomes the passing threshold for the future models' accuracy tests in the future portion of our project.

```
sampleSize <- floor(.7 * nrow(diabetes))
set.seed(125)
Ind <- sample(seq_len(nrow(diabetes)), size = sampleSize)

XTrain <- diabetes[Ind, 1:8]
XTest <- diabetes[-Ind, 1:8]

YTrain <- diabetes[Ind, 9]
YTest <- diabetes[-Ind, 9]
```

Now that we have our training and test sets, we can proceed to construct our classification models.

6. Random Forest

Random Forest is a tree-based classifier comprising of a collection of independent identically distributed random vectors and trees. Each tree votes for the most popular class, in our case, either NotDiabetic or Diabetic and the forest chooses the classification with the most votes.[6]

Our Random Forest model is created by using expand.grid() and train(). The mtry argument of expand.grid() is the number of predictor features available for splitting at each tree node. We choose two as the minimum and eight as the max to prevent overfitting. The train() runs through the training and test sets to build the tree, implements 10-fold cross-validation, and uses the ROC as the method to optimize the Random Forest model.

```
rf.expand <- expand.grid(mtry = 2:8)
set.seed(125)
diabetes.rf <- train(XTrain,
                     YTrain,
                     method = "rf",
                     metric = "ROC",
                     trControl = Folds,
                     tuneGrid = rf.expand)
diabetes.rf
```

```
## Random Forest
##
## 537 samples
##   8 predictor
##   2 classes: 'Diabetic', 'NotDiabetic'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 483, 483, 483, 483, 484, 484, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   2     0.8147057  0.5443137  0.8560159
##   3     0.8107381  0.5431373  0.8506587
##   4     0.8055560  0.5423856  0.8434286
##   5     0.8033849  0.5474183  0.8406111
##   6     0.8006177  0.5424183  0.8428810
##   7     0.7996720  0.5429412  0.8366746
##   8     0.7979395  0.5480719  0.8400794
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

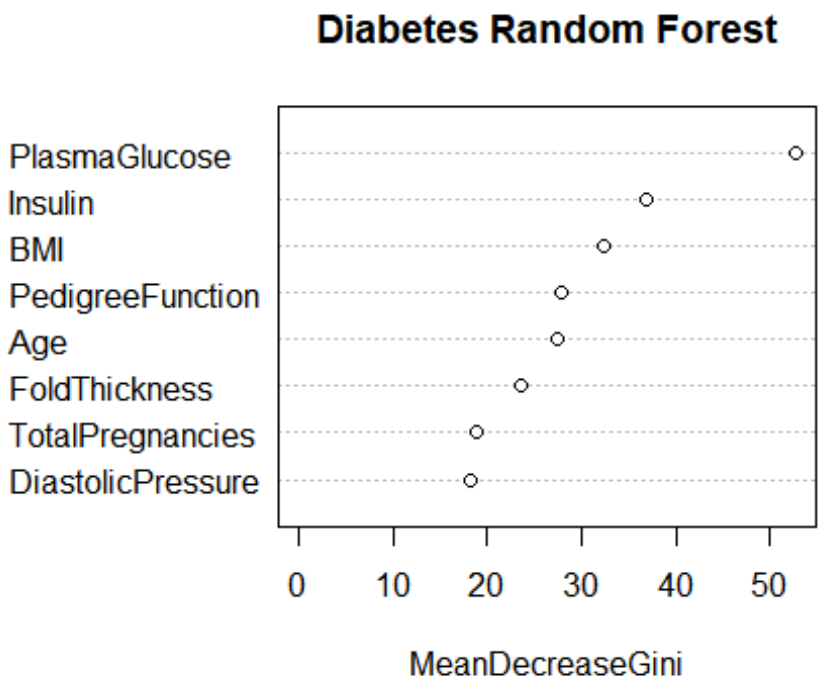
From this output, we can see the ROC is maximized when mtry = 2 and understand that Random Forest is optimal when two variables are split at each node.

Next, we will visualize the Random Forest model by plotting its variable of importance.

Variable of Importance

The variable of importance identifies the predictor with the most impact on classifying our class label, with the most impactful at the top and the least impactful at the bottom by the mean decrease in Gini coefficient. The mean decrease in Gini coefficient "measures how each variable contributes to the homogeneity of the nodes and leaves in the resulting random forest."[7]

```
varImpPlot(diabetes.rf$finalModel, main = "Diabetes Random Forest")
```



We can see our PlasmaGlucose predictor has the highest mean decrease in Gini coefficient and DiastolicPressure has the lowest mean decrease in Gini coefficient. Also, all predictors appear to be significant in classifying the Diagnosis response feature for our Random Forest model.

We will proceed from here to the SVM classification.

7. Support Vector Machine

A SVM is a supervised learning classifier that projects a binary class label on a hyperplane and separates the two classes of points by optimizing the margin. The data points on either side of the hyperplane are called support vectors and function as determining the margins between the class labels.[8] SVMs also have the parameters that impact its algorithm: C and gamma.

C is the trade-off between the classification of training examples and the margins and gamma is how far the influence of a single training example reaches. A small C maximizes the margin at the cost of accurately classifying training examples, and a large C ensures accurate classification of the training examples at the cost of maximizing the margin. A small gamma indicates each training example has a far reach from the margins and a large gamma indicates each training example has a close reach from the margins.[9] It is important to note that the LSVM only has the C parameter and the RSVM has both the C and gamma parameters.

For the context of this paper, we use the linear kernel and the radial basis function (RBF) kernel, a non-linear kernel, to project our data from the input space into a higher dimensional feature space because our dataset is not linearly separable.

7.1 Linear Support Vector Machine

We tune our LSVM by having the expand.grid() take on different cost values and then choose the C with the highest ROC. Then we use train() to run through the training and test sets to build the LSVM, implement 10-fold cross-validation, and use method = "svmLinear" for the linear kernel.

```
#Creates Linear SVM
linear.tune <- expand.grid(C = c(seq(.5, 5, by=.5)))
set.seed(125)
lsvm <- train(XTrain,
              YTrain,
              method = "svmLinear",
              metric = "ROC",
              trControl = Folds,
              tuneLength = 10,
              tuneGrid = linear.tune)
lsvm
```

```
## Support Vector Machines with Linear Kernel
##
## 537 samples
##   8 predictor
##   2 classes: 'Diabetic', 'NotDiabetic'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 483, 483, 483, 483, 484, 484, ...
## Resampling results across tuning parameters:
##
##   C    ROC        Sens       Spec
##   0.5  0.8280692  0.5313072  0.8995952
##   1.0  0.8279326  0.5307190  0.9009762
##   1.5  0.8279313  0.5318627  0.9006984
##   2.0  0.8279926  0.5313072  0.8998651
##   2.5  0.8279617  0.5296078  0.9034841
##   3.0  0.8280415  0.5296405  0.9021032
##   3.5  0.8279771  0.5301961  0.9023730
##   4.0  0.8280706  0.5329739  0.8995873
##   4.5  0.8280542  0.5307516  0.8998651
##   5.0  0.8281168  0.5296405  0.9020952
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was C = 5.
```

The optimal ROC gives us C = 5. We will now visualize this model to understand its margin and support vectors.

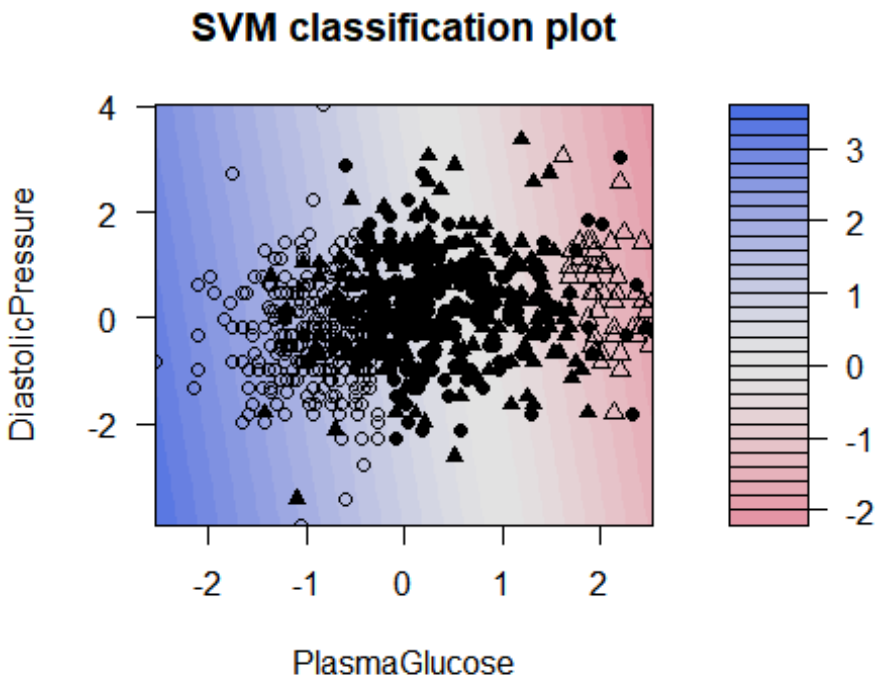
7.2 Linear Support Vector Machine Plot

With the information from the variable of importance, we will plot LSVM of the PlasmaGluccose predictor against the DiastolicPressure to visualize LSVM's fit with the ksvm() and plot(). The "vanilladot" argument of the ksvm() makes the model be fitted as a LSVM, and "C-svc" makes it a C-classification.

```
lsvm.fit = ksvm(Diagnosis~DiastolicPressure+PlasmaGlucose, data =
diabetes,type = 'C-svc', kernel = 'vanilladot')

##  Setting default kernel parameters

plot(lsvm.fit, data=diabetes)
```



The black filled shapes represent the support vectors. The triangles correspond to the training examples classified by DiastolicPressure. The circles correspond to the training examples classified by PlasmaGlucose. The red region indicates the Diabetic class label. The blue region represents the NotDiabetic class label.

The support vectors form a hyperplane that maximizes the margin for the NotDiabetic class label and minimizes the margin for the Diabetic class label. From looking at the shapes and colors, the training examples classified by the PlasmaGlucose tend to be NotDiabetic and the DiastolicPressure training examples lie closer within the Diabetic class. It is also

important to note that DiastolicPressure seems to have less training examples than PlasmaGlucose.

From here, we follow a similar process for the RSVM model.

7.3 Radial Support Vector Machine

We tune our RSVM by having the expand.grid() take on different cost and gamma (sigma in the code) values and then choose the C and gamma with the highest ROC. Then we use train() to run through the training and test sets to build the RSVM, implement 10-fold cross-validation, and use method = "svmRadial" for the RBF kernel.

```
radial.svm.expand <- expand.grid(sigma = c(2,3,4,5),
                                 C = c(.2,.4,.6,.8))
set.seed(125)
rsvm <- train(XTrain,
                    YTrain,
                    method = "svmRadial",
                    metric = "ROC",
                    trControl = Folds,
                    tuneGrid = radial.svm.expand)
rsvm
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 537 samples
##   8 predictor
##   2 classes: 'Diabetic', 'NotDiabetic'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 483, 483, 483, 483, 484, 484, ...
## Resampling results across tuning parameters:
##
##   sigma  C    ROC        Sens       Spec
##   2      0.2  0.7494299  0.3377451  0.8757937
##   2      0.4  0.7493828  0.3573529  0.8729683
##   2      0.6  0.7494299  0.3493137  0.8763571
##   2      0.8  0.7495542  0.3453922  0.8755238
##   3      0.2  0.7494299  0.3520588  0.8738254
##   3      0.4  0.7493828  0.3602614  0.8676905
##   3      0.6  0.7494299  0.3471895  0.8740794
##   3      0.8  0.7495542  0.3460784  0.8743889
##   4      0.2  0.7494299  0.3443137  0.8771508
##   4      0.4  0.7493828  0.3442484  0.8780238
##   4      0.6  0.7494299  0.3583333  0.8713333
##   4      0.8  0.7495542  0.3432026  0.8746667
##   5      0.2  0.7494299  0.3473203  0.8768889
##   5      0.4  0.7493828  0.3503268  0.8704762
##   5      0.6  0.7494299  0.3503922  0.8712937
```

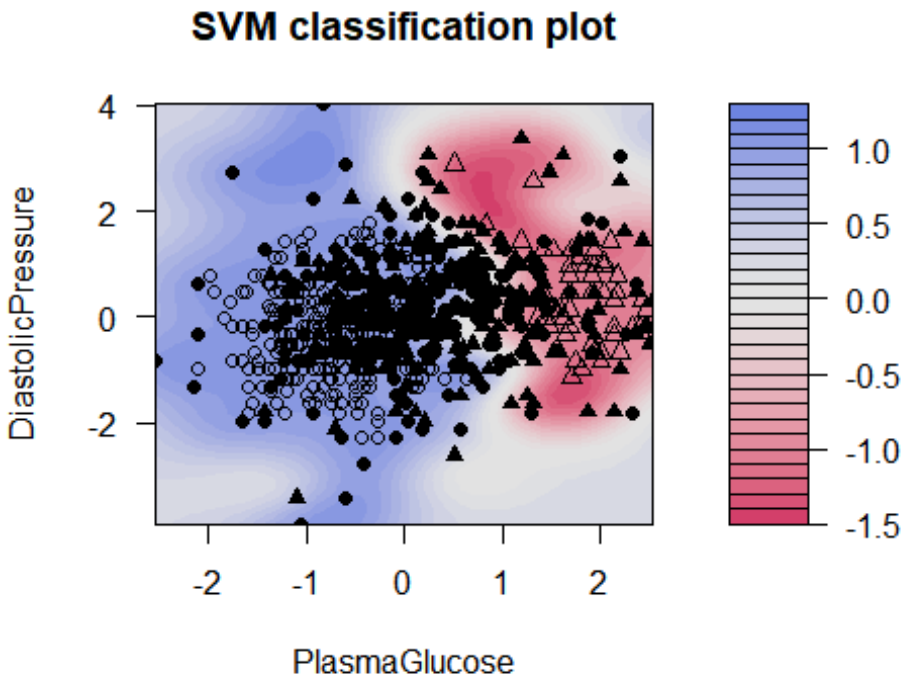
```
##   5      0.8  0.7495542  0.3505229  0.8752857
##
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were sigma = 5 and C = 0.8.
```

The optimal ROC gives us gamma = 5 and C = 0.8. We will now visualize this model to understand its margin and support vectors.

7.4 Radial Support Vector Machine Plot

Once again, we are using the information from the variable of importance to plot the RSVM. This time the ksvm() takes the argument "rbfdot" to represent the RBF kernel.

```
rsvm.fit = ksvm(Diagnosis~DiastolicPressure+PlasmaGlucose, data =
diabetes,type = 'C-svc', kernel = 'rbfdot')
plot(rsvm.fit, data=diabetes)
```



Similarly to the LSVM plot, DiastolicPressure has less training examples than PlasmaGlucose and most of PlasmaGlucose's training examples appear to be classified as NotDiabetic. Also, its hyperplane maximizes the PlasmaGlucose training examples many more than the LSVM, which could negatively impact its classification accuracy. With respect to the support vectors, they appear more spread out and forms a hyperplane that segments the class labels into separate groups.

Knowing this about each of our three classification machine learning algorithms, we can carry out with testing their accuracy.

8. Testing Model Accuracy

We have reached the point in our paper where will can test the accuracy of our models by using our test set to see if the model surpasses the 65% accuracy threshold. We determine the accuracy by forming a confusion matrix, a contingency table that explains the performance of an algorithm's through its predicted and actual values.[10] The confusion matrix for each model can easily be constructed with the confusionMatrix().

8.1 Random Forest Accuracy

```
confusionMatrix(diabetes.rf)
```

```
## Cross-Validated (10 fold, repeated 10 times) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##              Reference
## Prediction    Diabetic NotDiabetic
##   Diabetic        18.1         9.6
##   NotDiabetic     15.2        57.1
##
##  Accuracy (average) : 0.752
```

8.2 Linear Support Vector Machine Accuracy

```
confusionMatrix(lsvm)
```

```
## Cross-Validated (10 fold, repeated 10 times) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##              Reference
## Prediction    Diabetic NotDiabetic
##   Diabetic        17.7         6.5
##   NotDiabetic     15.7        60.1
##
##  Accuracy (average) : 0.7778
```

8.3 Radial Support Vector Machine Accuracy

```
confusionMatrix(rsvm)
```

```
## Cross-Validated (10 fold, repeated 10 times) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##              Reference
## Prediction    Diabetic NotDiabetic
##   Diabetic        11.7         8.3
##   NotDiabetic     21.7        58.3
##
##  Accuracy (average) : 0.7002
```

All three models pass the 65% accuracy threshold and LSVM has the highest accuracy, making it the best model to classify diabetes in Pima Native American Women.

9. Conclusion

Random Forest, LSVM, AND RSVM passed the 65% accuracy threshold and proved that these are useful algorithms in understanding medical and biological data. These models could improve their accuracy and be more powerful through better tuning and a dataset with less missing values. Nevertheless, these models are important tools to diagnose diabetes in communities, to diagnose other diseases, and even classify genetic mutations to impact medical research. This paper functions as proof that machine learning is a valuable asset and has a wide range of applications in the medical field.

10. References

1. Narayan, Venkat. "Diabetes Mellitus in Native Americans: The Problem and Its Implications." Changing Numbers, Changing Needs: American Indian Demography and Public Health., U.S. National Library of Medicine, 1 Jan. 1996, www.ncbi.nlm.nih.gov/books/NBK233089/.
2. Sigillito, Vincent . "Pima Indians Diabetes Data Set ." UCI Machine Learning Repository: Data Set, 5 Sept. 1990, archive.ics.uci.edu/ml/datasets/Pima Indians Diabetes.
3. "How Do I Perform Multiple Imputation Using Predictive Mean Matching in R? | R FAQ." IDRE Stats, stats.idre.ucla.edu/r/faq/how-do-i-perform-multiple-imputation-using-predictive-mean-matching-in-r/.
4. Thornton, Chris. "Machine Learning - Lecture 5: Cross-Validation." University of Sussex, users.sussex.ac.uk/~christ/crs/ml/lec03a.html.
5. Brownlee, Jason. "What Is the Difference Between Test and Validation Datasets?" Machine Learning Mastery, 26 July 2017, machinelearningmastery.com/difference-test-validation-datasets/.
6. Breiman, Leo. "Random Forests." Machine Learning, vol. 45, no. 1, 2001, pp. 5-32., doi:https://doi.org/10.1023/A:1010933404324.
7. Dinsdale, Liz, and Rob Edwards. "Metagenomics. Statistics." Dinsdale Et Al. Supplemental Material, San Diego State University, dinsdalelab.sdsu.edu/metag.stats/code/randomforest.html.
8. Patel, Savan. "Chapter 2 : SVM (Support Vector Machine) - Theory ." Medium, Machine Learning 101, 3 May 2017, medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72.
9. Berwick, Robert. "An Idiot's Guide to Support Vector Machines (SVMs)." MIT Machine Learning for Big Data and Text Processing, web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf.
10. Powers , David M W. "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation ."

www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf.

11. Software

1. https://www.r-project.org/
2. https://www.rstudio.com/
3. https://cran.r-project.org/web/packages/corrplot/index.html
4. https://cran.r-project.org/web/packages/e1071/index.html
5. https://cran.r-project.org/web/packages/RSNNS/index.html
6. https://cran.r-project.org/web/packages/caret/index.htmll
7. https://cran.r-project.org/web/packages/AppliedPredictiveModeling/index.html
8. https://cran.r-project.org/web/packages/mice/index.html
9. https://cran.r-project.org/web/packages/randomForest/index.html
10. https://cran.r-project.org/web/packages/VIM/index.html
11. https://cran.r-project.org/web/packages/kernlab/index.html

12. Appendix

```
knitr::opts_chunk$set(echo = TRUE)
#Accessing packages for this project
library(corrplot)
library(e1071)
library(RSNNS)
library(randomForest)
library(mice)
library(kernlab)
library(caret)
library(AppliedPredictiveModeling)
library(VIM)
#Create a vector of the feature names
headers <- c("TotalPregnancies", "PlasmaGlucose", "DiastolicPressure",
"FoldThickness",
"Insulin", "BMI", "PedigreeFunction", "Age", "Diagnosis")

#Import data
url <- paste0("http://archive.ics.uci.edu/ml/machine-learning-databases/",
"pima-indians-diabetes/pima-indians-diabetes.data")
diabetes <- read.csv(url(url),header = FALSE, col.names = headers)
str(diabetes)
diabetes$Diagnosis <- as.factor(ifelse(diabetes$Diagnosis == 0,
"NotDiabetic", "Diabetic"))
pairs(diabetes)
#Creates loops to make all 0's NA
for (i in 2:6){
  for (n in 1:nrow(diabetes)){
```

```r
    if (diabetes[n, i] == 0){
      diabetes[n, i] <- NA
    }
  }
}
table(is.na(diabetes))
aggr(diabetes[,2:6], cex.lab=1, cex.axis = .4, numbers = T, gap = 0)
#visualises a scatterplot of missing values
scattmatrixMiss(diabetes)
com.diabetes <- mice(diabetes, m = 3, method = 'pmm', seed = 125)
#Density plot original vs imputed dataset
densityplot(com.diabetes)
diabetes <- complete(com.diabetes)
corrplot(cor(diabetes[,-9]),type = "lower", method = "number")
diabetes[, 1:8] <- scale(diabetes[, 1:8], center = TRUE, scale = TRUE)
Folds <- trainControl(method = "repeatedcv",
                      number = 10,
                      repeats = 10,
                      classProbs=TRUE,
                      summaryFunction=twoClassSummary)
prop.table(table(diabetes$Diagnosis))
sampleSize <- floor(.7 * nrow(diabetes))
set.seed(125)
Ind <- sample(seq_len(nrow(diabetes)), size = sampleSize)

XTrain <- diabetes[Ind, 1:8]
XTest <- diabetes[-Ind, 1:8]

YTrain <- diabetes[Ind, 9]
YTest <- diabetes[-Ind, 9]
rf.expand <- expand.grid(mtry = 2:8)
set.seed(125)
diabetes.rf <- train(XTrain,
                      YTrain,
                      method = "rf",
                      metric = "ROC",
                      trControl = Folds,
                      tuneGrid = rf.expand)
diabetes.rf
varImpPlot(diabetes.rf$finalModel, main = "Diabetes Random Forest")
#Creates Linear SVM
linear.tune <- expand.grid(C = c(seq(.5, 5, by=.5)))
set.seed(125)
lsvm <- train(XTrain,
              YTrain,
              method = "svmLinear",
              metric = "ROC",
              trControl = Folds,
              tuneLength = 10,
              tuneGrid = linear.tune)
```

```
lsvm
lsvm.fit = ksvm(Diagnosis~DiastolicPressure+PlasmaGlucose, data =
diabetes,type = 'C-svc', kernel = 'vanilladot')
plot(lsvm.fit, data=diabetes)
radial.svm.expand <- expand.grid(sigma = c(2,3,4,5),
                                 C = c(.2,.4,.6,.8))
set.seed(125)
rsvm <- train(XTrain,
                    YTrain,
                    method = "svmRadial",
                    metric = "ROC",
                    trControl = Folds,
                    tuneGrid = radial.svm.expand)
rsvm
rsvm.fit = ksvm(Diagnosis~DiastolicPressure+PlasmaGlucose, data =
diabetes,type = 'C-svc', kernel = 'rbfdot')
plot(rsvm.fit, data=diabetes)
confusionMatrix(diabetes.rf)
confusionMatrix(lsvm)
confusionMatrix(rsvm)
```