Project

Chris Meade

12/7/2016

Contents

Abstract

Diabetes occurs more often in the Pima Indian tribe of southern Arizona than any other population of people in the world.(1) By selecting the optimal parameters for three machine learning algorithms – Random Forest decision trees, a Support Vector Machine (SVM), and a Multilayer Perceptron (MLP) artifical neural network (ANN) – this paper proves that diabetes can be accurately diagnosed in Pima women. These results indicate that machine learning can be a valuable tool for doctors to help with the diagnosis of diseases.

1. Introduction

This paper explores the viability of the application machine learning to the domain of medicine. Can statistical learning algorithms be utilized by doctors to help make accurate diagnoses and aid in decision making as it pertains to patient health? To attempt to answer this question, we train three distinct machine learning algorithms – Random Forest decision trees, a Support Vector Machine (SVM), and a Multilayer Perceptron (MLP) artificial neural network – using data about diabetes occurances in Pima Indian Women. All analysis was performed with the R statistical computing software package and associated machine learning libraries in the RStudio integrated development enviornment. Our results clearly indicate that all three algorithms had a high success rate in correctly diagnosing diabetes in Pima women. This lends support to the proposed hypothesis that machine learning does have a place in doctors' arsenal.

1.1 About the Data

The data about diabetes incidence in Pima women used in our analysis are freely avaiable from the University of California, Irvine Machine Learning Repository at http://archive.ics.uci.edu/ml/. This dataset was created to try to predict the onset of diabetes in women belonging to the Pima Indian Tribe. This dataset contains 768 observations of nine features. The features in the order they appear in the dataset are as follows:

Feature Description	Feature name in dataset
Number of times pregnant	timePregnant
Plasma glucose concentration at 2 hours in an oral glucose tolerance test	plasmaGlucose
Diastolic blood pressure (mm Hg)	diastolicPressure
Triceps skin fold thickness (mm)	tricepThickness
2-Hour serum insulin (mu U/ml)	serumInsulin
Body mass index (weight in kg/(height in m)^2)	bmi
Diabetes pedigree function	pedigreeFunction
Age (years)	age
Class variable (diabetic or not diabetic)	diabetes

A full description of this dataset is available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2245318/

1.2 Software Used in Analysis

All analysis was performed with the R statistical computing environment. The functionality of R was extended by importing the following software libraries: `caret`, `mice`, `randomForest`, `kernlab` `VIM`, and `RSNNS`. These librarys add support for data imputation and for the three machine learning algorithms used in our analysis.

The analysis was written in the RStudio integrated development enviornment. This report was generated using the 'rmarkdown' R package.

1.3 Outline of Analysis

We begin by loading data into R and conducting some exploratory analysis. We then impute missing values. Next, the data is standardized and split into a training and test set. Afterwards, we begin constructing models and optimizing paramters. Finally, we assess the classification accuracy of our three models on the test set to choose the best one.

2. Data Processing

2.1 Data Aquisition

We must first import our data into R and load the required R packages used in our analysis.

```
library(caret)
library(mice)
library(randomForest)
library(kernlab)
library(RSNNS)
library(VIM)

#Create a vector of the feature names
headers <- c("timesPregnant", "plasmaGlucose", "diastolicPressure", "tricepThickness",
             "serumInsulin", "bmi", "pedigreeFunction", "age", "diabetes")

#Import data
www <- paste0("http://archive.ics.uci.edu/ml/machine-learning-databases/",
              "pima-indians-diabetes/pima-indians-diabetes.data")

data <- read.csv(url(www),
                 header = FALSE,
                 col.names = headers)
```

We examine the structure of our data table to ensure that it was imported correctly.

```
str(data)
```

```
## 'data.frame':    768 obs. of  9 variables:
##  $ timesPregnant    : 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 ...
##  $ tricepThickness  : int  35 29 0 23 35 0 32 0 45 0 ...
##  $ serumInsulin     : 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 ...
##  $ diabetes         : int  1 0 1 0 1 0 1 0 1 1 ...
```

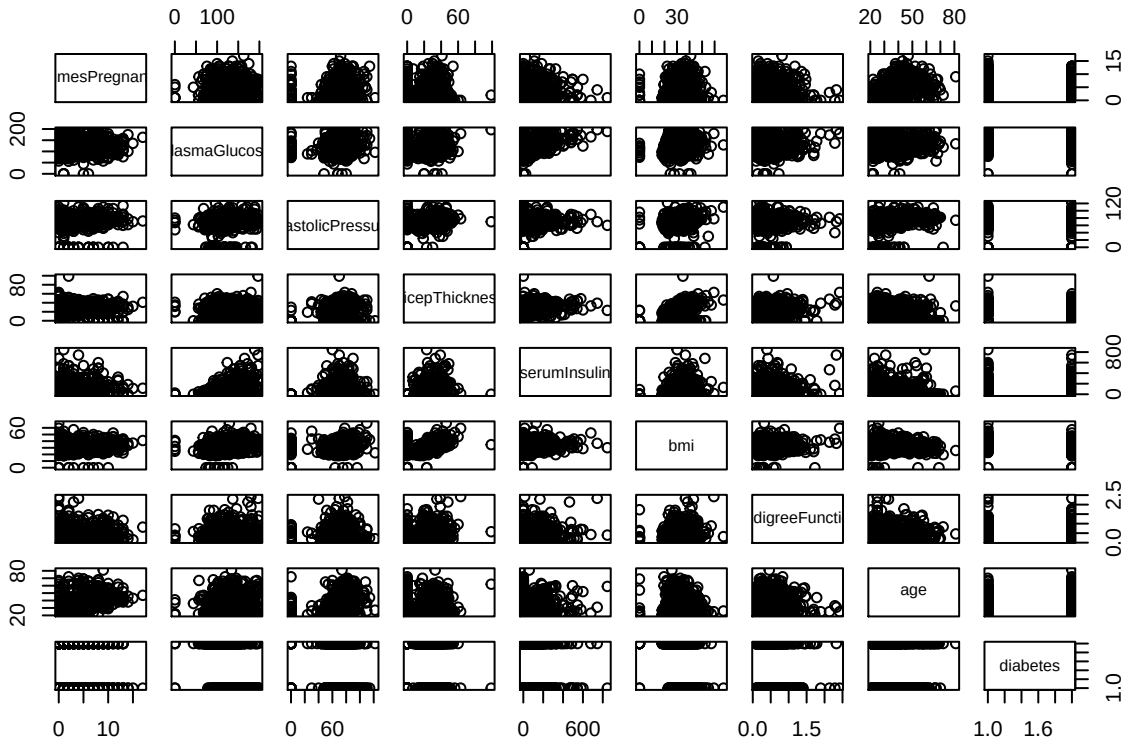
Everything appear to be in order, so next we encode the class label of the `diabetes` feature to be more reable by changing `0` to `notDiabetic` and `1` to `Diabetic`. We then covert this feature to a factor.

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

2.2 Exploratory Analysis

Now that we have loaded our data and have imported all required software libraries, we perform some exploratory analysis. We begin by constructing a scatterplot matrix.

```
pairs(data)
```



From this scatterplot matrix we observe that several features have `0` valued observation.

2.3 Missing Data

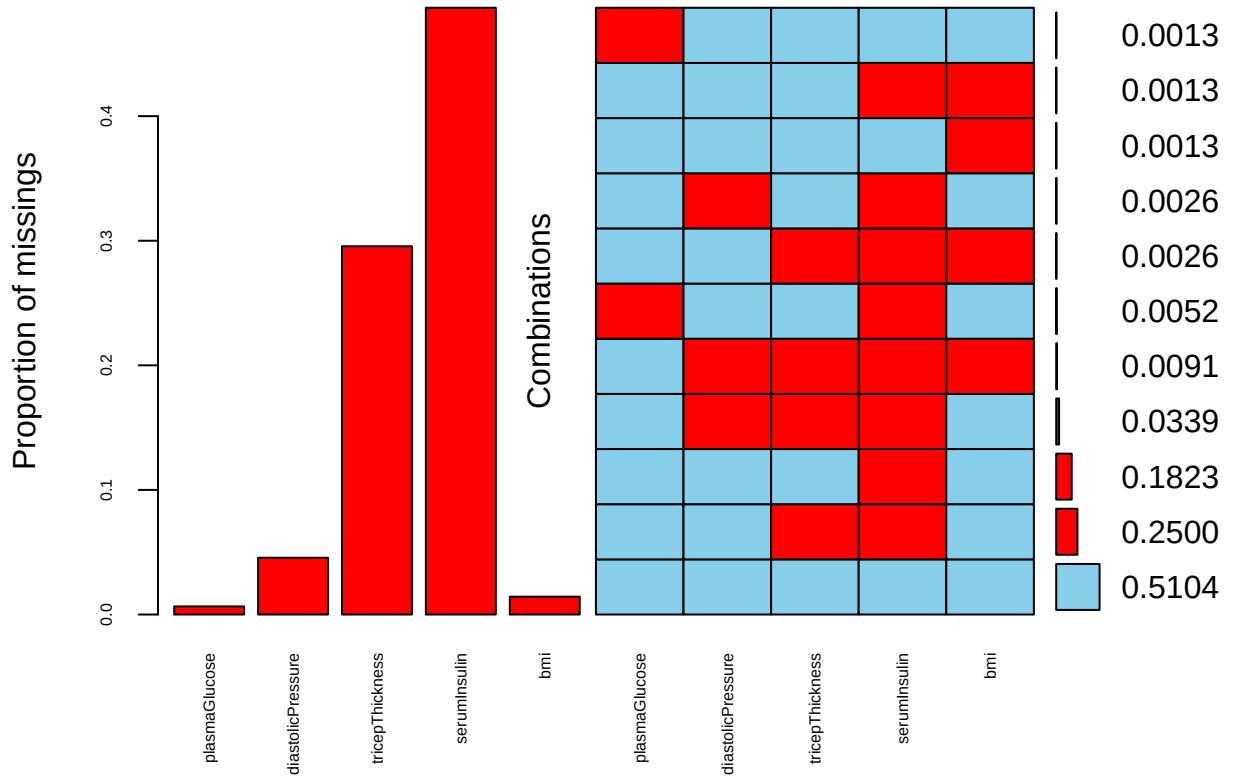
From our exploratory analysis, we found that several features contain `0` valued observations. For several of the features, a `0` value is biologically impossible, namely in `plasamaGlucose`, `diastolicPressure`, `tricepThickness`, `serumInsulin`, `bmi`. We conclude that although the dataset does not explicitly contain missing values, it does contain implicity missing values encoded as `0`s.

We decide to explictily encode the missing values in the five listed features as `NA`s.

```
for (i in 2:6){
  for (n in 1:nrow(data)){
    if (data[n, i] == 0){
      data[n, i] <- NA
    }
  }
}
```

Now that missing data is correcty encoded as `NA`s, we construct an aggregation plot to count the precise number of missing values.

```
aggr(data[,2:6], cex.lab=1, cex.axis = .5, numbers = T, gap = 0)
```



The left plot shows the proportion of missing values to total observations for each feature. We note that over half of all observations for `serumInsulin` and nearly a third `tricepThickness` observations are missing.

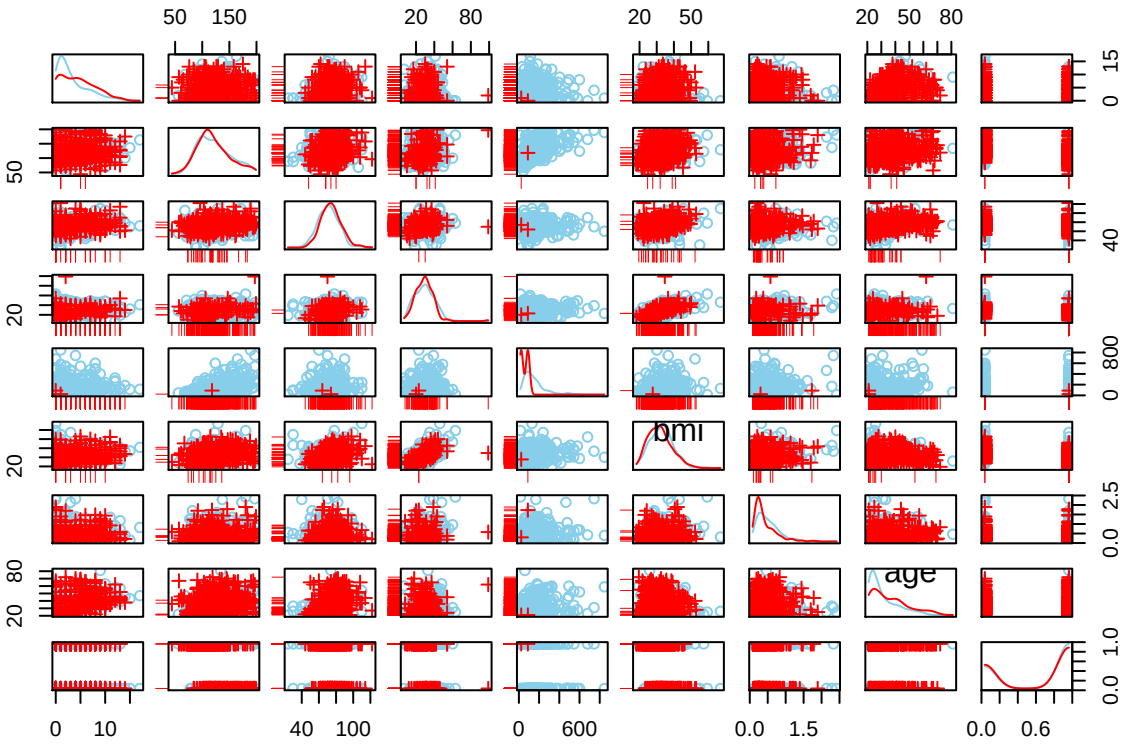
The plot on the right indicates that only a little over half of the observations are complete. Due to the number of observations with missing values, we choose not to drop these observations from our analysis. Instead, we will impute missing values using a technique called Imputation by Predicted Mean matching, which "imputes missing values by means of the nearest-neighbor donor with distance based on the expected values of the missing variables conditional on the observed covariates."(2)

Imputation by Predicted Mean Matching is advantagous over other methods of data imputation because it "can provide valid inference when data are missing at random." (3)

This introduces an assumption: that data are at least missing at random. That is, we must be sure that our data are not Missing Not at Random (MNAR).

To test if this is a reasonable assumption, we construct a scatterplot matrix highlighting missing values.

```
scattmatrixMiss(data)
```



We note no discernible relationship between variables and missing values, so our assumption is valid. We proceed with the Imputation by Predictive Mean Matching.

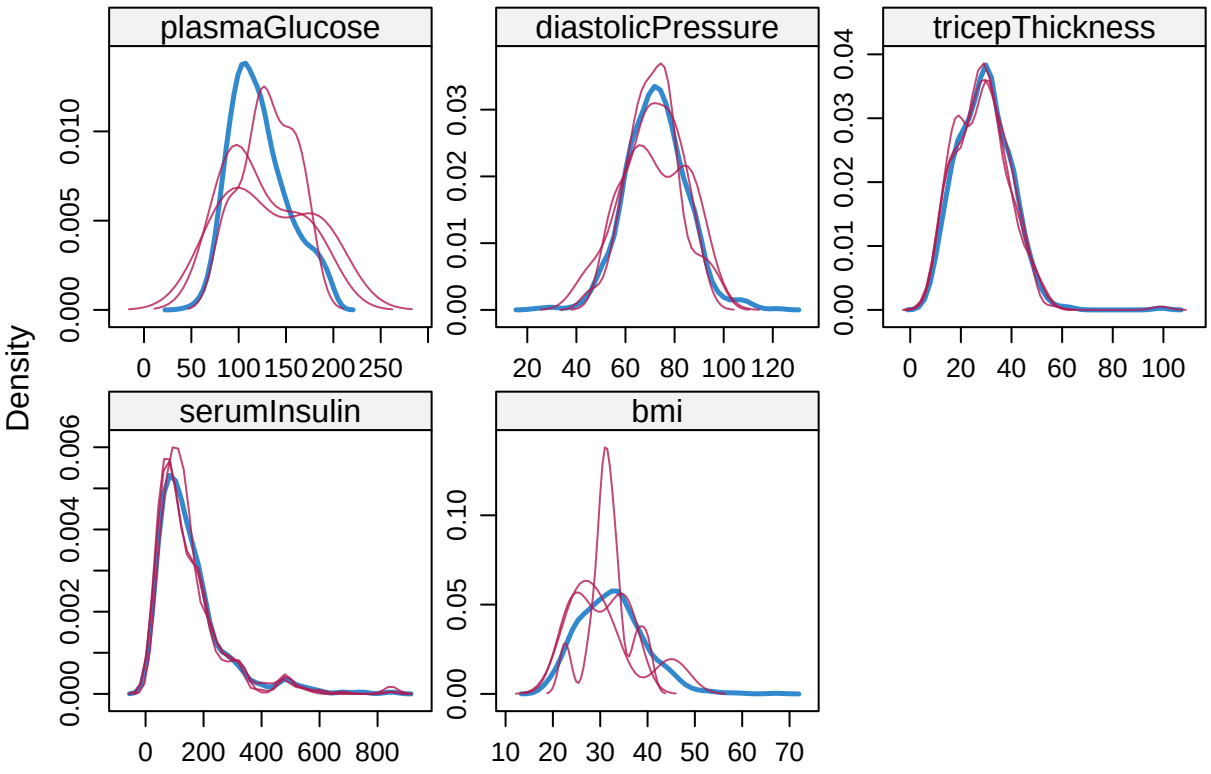
```
tempdata <- mice(data, m = 3, method = 'pmm', seed = 100)
```

```
##
##  iter imp variable
##    1   1  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    1   2  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    1   3  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    2   1  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    2   2  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    2   3  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    3   1  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    3   2  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    3   3  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    4   1  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    4   2  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    4   3  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    5   1  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    5   2  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
##    5   3  plasmaGlucose  diastolicPressure  tricepThickness  serumInsulin  bmi
```

```
data <- complete(tempdata)
```

We examine the distributions of our imputed data (red) compared to the original data (blue).

```
densityplot(tempdata)
```



The distributions are approximately equal, so we may proceed to feature selection.

2.4 Feature Selection

We ensure that no two features are too highly correlated by constructing a correlation matrix. If any two features have a correlation coefficient greater than `0.7`, we will consider removing one of them.

```
correlationMatrix <- cor(data[,1:8])
findCorrelation(correlationMatrix, cutoff=0.7)
```

```
## integer(0)
```

This is not necessary, though, as no two features are that highly correlated. Consequently, we choose not to eliminate any features from our analysis.

2.5 Data Standardization

Since the features of the dataset represent different units, we choose to standardize the data but making the mean of each column `0` with a standard deviation of `1`.

This is accomplished with a very simple algorithm:

Replace each observation of a random variable with the output of the following function:

$$f(x_i) = \frac{x_i - mean(X)}{sd(X)}$$

Then repeat for each random variable.

This standardization is easily performed with R.

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

3. Training the Algorithms

With our exploratory analysis and data processing complete, we begin building models. Models for each of our three methods will be constructed from a training set of observations using 10-fold cross validation to prevent overfitting. This is easily implemented using the `caret` package.

```
tenFoldCV <- trainControl(method = "repeatedcv",
                          number = 10,
                          repeats = 10,
                          classProbs=TRUE,
                          summaryFunction=twoClassSummary)
```

The best model for each of the three methods will then be selected by its area under the ROC curve. This score, $x \in [0, 1]$, represents model accuracy. A score close to 1 indicates high accuracy.

3.1 Predicting the most common label

Of the 768 observations, about 65% have class label 'notDiabetic' and the remaining 35% have class label 'Diabetic'. So by predicting 'notDiabetic' for every class, one will already be 65% accurate. Thus any model must have an accuracy greater than 65% to be viable.

3.2 Training and Test Sets

We use 70% of the observations to train the models and reserve the remaining 30% to test performance.

```
sampleSize <- floor(.7 * nrow(data))

set.seed(131)
trainIndices <- sample(seq_len(nrow(data)), size = sampleSize)

x.train <- data[trainIndices, 1:8]
y.train <- data[trainIndices, 9]

x.test <- data[-trainIndices, 1:8]
y.test <- data[-trainIndices, 9]
```

3.3 Random Forest

This algorithm presents a tree-based classification method. It is unique from other tree-based methods because whenever a split in the tree is considered, the algorithm may only use a random subset of predictor variables as split candidates. Then only one predictor in the random subset is used in the split.(4)

We train the model with the following function, which creates cross-validated models for each different possible size, called `mtry`, of the random sample. We exclude the possibility of `mtry = 1` to prevent overfitting the data.

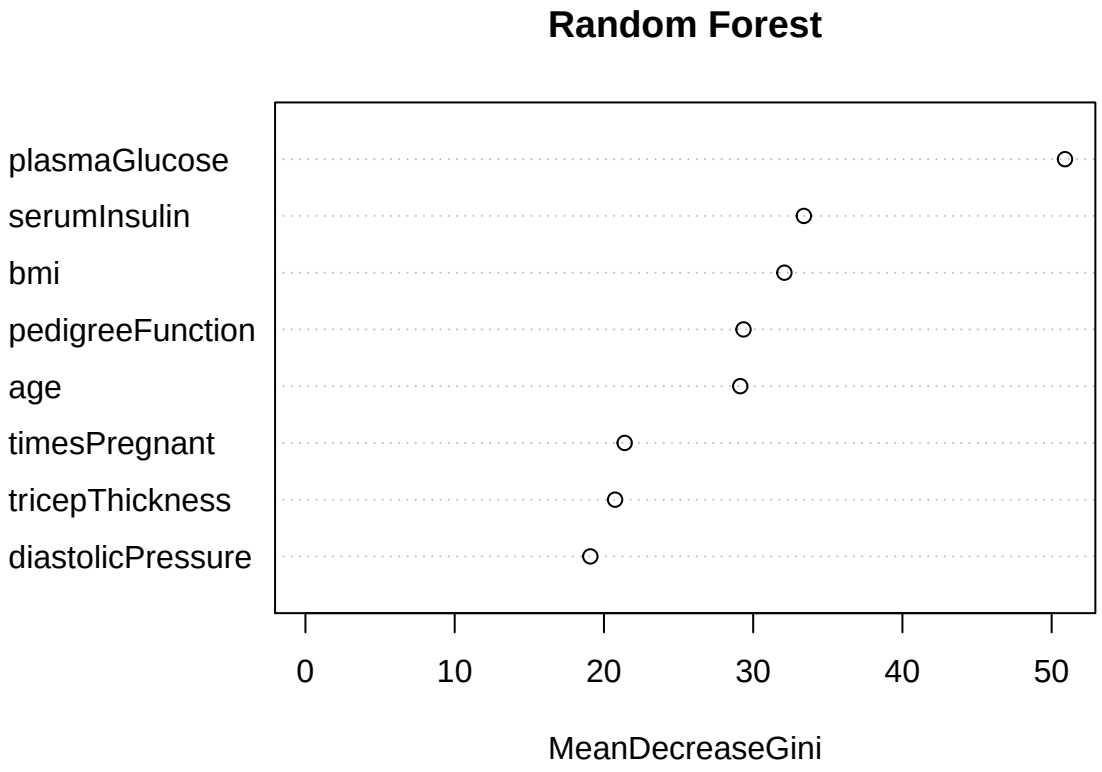
```r
rf.expand <- expand.grid(mtry = 2:8)

set.seed(100)
rf <- caret::train(x.train,
                   y.train,
                   method = "rf",
                   metric = "ROC",
                   trControl = tenFoldCV,
                   tuneGrid = rf.expand)

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, 484, 483, 483, 483, 483, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   2     0.8089501  0.5339542  0.8667568
##   3     0.8047676  0.5350327  0.8670120
##   4     0.8033523  0.5407190  0.8601051
##   5     0.8020815  0.5413072  0.8595495
##   6     0.8005601  0.5343464  0.8581456
##   7     0.8002150  0.5349020  0.8550901
##   8     0.7995324  0.5377451  0.8492643
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

The function trained several different models with seven different possible random subset sizes. The highest area under the ROC curve was achieved when only 2 randomly chosed predictors were considered.

Next, we examine the importance of all the variables.

```r
varImpPlot(rf$finalModel, type = 2, main = "Random Forest")
```



The plot lists the importance of any given variable to the the model, with the most important variable at the top and least important variable at the bottom.(5) Since there is only a single jump after `plasmaGlucose`, all variables appear to be significant to the model. We conclude that it is not necessary to eliminate variables or retrain the model.

Thus the random forest with `mtry = 2` from above is chosen as our final model.

3.4 Support Vector Machine

The Support Vector Machine, generally speaking, is a classifcation method that attempts to linearly separate observations based on class label. This dataset was not linearly separable, so we use a function kernel to tranform our data into a higher dimensional feature space where it is more easily separable by a hyperplane.

Several such transformations exist, but for brevity we choose to compare only two prominent ones – a linear kernel and a radial basis funtion kernel.

We begin by training the SVM using a linear kernel and 10-fold cross validation.

```
linear.svm.expand <- expand.grid(C = c(.1, 1, 10))
set.seed(131)
linear.svm <- caret::train(x.train,
                           y.train,
                           method = "svmLinear",
                           metric = "ROC",
                           trControl = tenFoldCV,
                           tuneLength = 10,
                           tuneGrid = linear.svm.expand)
linear.svm
```

```
## 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: 484, 483, 483, 484, 484, 483, ...
## Resampling results across tuning parameters:
##
##   C     ROC        Sens       Spec
##    0.1  0.8337570  0.5256209  0.8922748
##    1.0  0.8311587  0.5222876  0.8986562
##   10.0  0.8315109  0.5250654  0.8986562
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was C = 0.1.
```

The Linear Kernel Support Vector Machine has a single parameter, cost, denoted `C`. We first broadly try to find the optimal value of `C` by training models for `C = .1, 1, and 10`. After finding that `C = .1` produces the best model, we narrow in our search.

```
linear.svm.expand2 <- expand.grid(C = c(.05, .1, .15))
set.seed(131)
linear.svm2 <- caret::train(x.train,
                            y.train,
                            method = "svmLinear",
                            metric = "ROC",
                            trControl = tenFoldCV,
                            tuneGrid = linear.svm.expand2)
linear.svm2
```

```
## 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: 484, 483, 483, 484, 484, 483, ...
## Resampling results across tuning parameters:
##
##   C     ROC        Sens       Spec
##   0.05  0.8360782  0.5355229  0.8867492
##   0.10  0.8337570  0.5250980  0.8917267
##   0.15  0.8317649  0.5233660  0.8925450
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was C = 0.05.
```

Area under the ROC curve was maximized by the model with `C = 0.05`. We will now construct a SVM with a radial basis function kernel to compare results. This model introduces a new parameter, sigma, which controls model slack. Just as before, we begin by broadly narrowing in optimal parameter values.

```
radial.svm.expand <- expand.grid(sigma = c(.2, .4, .6, .8),
                                 C = c(.1, 1, 5, 10, 100))
set.seed(131)
radial.svm <- caret::train(x.train,
                           y.train,
                           method = "svmRadial",
                           metric = "ROC",
                           trControl = tenFoldCV,
                           tuneGrid = radial.svm.expand)

radial.svm
```

```
## 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: 484, 483, 483, 484, 484, 483, ...
## Resampling results across tuning parameters:
##
##   sigma  C      ROC        Sens       Spec
##   0.2      0.1  0.8143866  0.5988562  0.8397297
##   0.2      1.0  0.8058166  0.4854575  0.8953904
##   0.2      5.0  0.7652529  0.4204248  0.8775826
##   0.2     10.0  0.7526302  0.3832026  0.8751652
##   0.2    100.0  0.7299535  0.3176144  0.8806682
##   0.4      0.1  0.8006421  0.5346078  0.8735210
##   0.4      1.0  0.7906976  0.4601961  0.8940015
##   0.4      5.0  0.7502196  0.3725163  0.8792943
##   0.4     10.0  0.7384676  0.3398693  0.8773649
##   0.4    100.0  0.7351139  0.3142810  0.8723498
##   0.6      0.1  0.7850177  0.4733333  0.8796171
##   0.6      1.0  0.7800271  0.4175163  0.8923048
##   0.6      5.0  0.7392252  0.3421569  0.8740165
##   0.6     10.0  0.7367109  0.3216340  0.8756982
##   0.6    100.0  0.7350554  0.3217320  0.8731982
##   0.8      0.1  0.7704077  0.4233987  0.8845646
##   0.8      1.0  0.7700430  0.3938235  0.8939640
##   0.8      5.0  0.7357902  0.3086275  0.8787538
##   0.8     10.0  0.7344897  0.3088235  0.8762387
##   0.8    100.0  0.7344920  0.3103595  0.8798574
##
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were sigma = 0.2 and C = 0.1.
```

We now narrow our search grid to values in the neighborhoods of `sigma = 0.2` and `C = .1` and construct more models.

```
radial.svm.expand2 <- expand.grid(sigma = c(.15, .2, .25),
                                  C = c(.01, .05, .1, .15, .25))
set.seed(131)
radial.svm2 <- caret::train(x.train,
                            y.train,
                            method = "svmRadial",
                            metric = "ROC",
                            trControl = tenFoldCV,
                            tuneGrid = radial.svm.expand2)

radial.svm2
```

```
## 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: 484, 483, 483, 484, 484, 483, ...
## Resampling results across tuning parameters:
##
##   sigma  C     ROC        Sens       Spec
##   0.15   0.01  0.8175172  0.6050980  0.8322222
##   0.15   0.05  0.8175799  0.6090850  0.8316517
##   0.15   0.10  0.8175790  0.6142810  0.8305781
##   0.15   0.15  0.8170617  0.5727778  0.8546622
##   0.15   0.25  0.8154163  0.5222876  0.8842943
##   0.20   0.01  0.8143094  0.6005556  0.8380706
##   0.20   0.05  0.8144675  0.6012745  0.8380556
##   0.20   0.10  0.8143866  0.6006536  0.8397147
##   0.20   0.15  0.8143630  0.5914052  0.8466441
##   0.20   0.25  0.8123248  0.5241176  0.8812613
##   0.25   0.01  0.8112813  0.5858170  0.8502703
##   0.25   0.05  0.8110136  0.5836601  0.8513814
##   0.25   0.10  0.8111698  0.5863072  0.8480255
##   0.25   0.15  0.8110767  0.5842810  0.8469144
##   0.25   0.25  0.8104641  0.5324837  0.8748799
##
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were sigma = 0.15 and C = 0.05.
```

So the final SVM model using the radial bias function kernel achieves a maximal area under the ROC curve value when `sigma = .15` and `C = 0.01`. Since this AUC is clearly below the AUC achieved with the linear kernel, we choose the SVM with the linear kernel transformation and `C = 0.05` as our final Support Vector Machine model.

3.5 Multilayer Perceptron Artifical Neural Network

This type of model works by feeding information through a series of layers, each consisting of artificial neurons which weight the inputs and allow the model to come to a final conclusion. Training begins at input layer.

Then information is passed in to a given number of hidden layers. The number of hidden layers is the model parameter we will optimize. Finally, an output layer relays the final decision about an observation's class label.(6)

```
set.seed(131)
mlpnn <- caret::train(x.train,
                      y.train,
                      method = "mlpML",
                      metric= "ROC",
                      trControl=tenFoldCV)

mlpnn
```

```
## Multi-Layer Perceptron, with multiple layers
##
## 537 samples
##   8 predictor
##   2 classes: 'Diabetic', 'notDiabetic'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 484, 483, 483, 484, 484, 483, ...
## Resampling results across tuning parameters:
##
##   layer1  ROC        Sens       Spec
##   1       0.8275189  0.6099346  0.8214865
##   3       0.8094735  0.5471569  0.8566141
##   5       0.8042522  0.5602941  0.8421997
##
## Tuning parameter 'layer2' was held constant at a value of 0
##
## Tuning parameter 'layer3' was held constant at a value of 0
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were layer1 = 1, layer2 = 0 and
##  layer3 = 0.
```

Performance is maximized with one hidden layer, so we choose this one as our final model for this method.

Selecting the Best Overall Model

We now have three final models, one for each method. Of these three, the model will the highest classification accuracy on our test set will be selected as the best model for this problem.

```
#RF Test accuracy
rf.predict <- predict(rf$finalModel, x.test)
rf.test.accuracy <- mean(rf.predict == y.test)
rf.test.accuracy
```

```
## [1] 0.7835498
```

```r
#SVM Test accuracy
svm.predict <- predict(linear.svm2$finalModel, x.test)
svm.test.accuracy <- mean(svm.predict == y.test)
svm.test.accuracy
```

```
## [1] 0.7662338
```

```r
#MLPNN Test Accuracy
mlpnn.predict <- predict(mlpnn$finalModel, x.test)
mlpnn.predict <- as.data.frame(mlpnn.predict)
mlpnn.predict$prediction <- ifelse(mlpnn.predict$V1 >= .5, "Diabetic", "notDiabetic")
mlpnn.test.accuracy <- mean(mlpnn.predict$prediction == y.test)
mlpnn.test.accuracy
```

```
## [1] 0.7402597
```

The MLP artificial neural network achieves the greatest classification accuracy, so we choose it as the best model.

Conclusion

With a very small and incomplete training set, data mining proved to be a viable method for diabetes diagnosis in Pima Indian women. Of the three algorithms tested, all performed better than the minimum viable accuracy rate. The multilayer perceptron artificial neural network algorithm performed best of all, and was consequently chosen as the best model for this problem. We believe that performance could be further improved with the availability of more data. This data would also ideally be free of missing values. Despite these hinderances, machine learning proved itself to be a valuable contribution to the medical industry.

References

(1) http://diabetes.diabetesjournals.org/content/53/5/1181
(2) http://www.stefvanbuuren.nl/publications/2014%20Semicontinuous%20-%20Stat%20Neerl.pdf
(3) http://bmcmedresmethodol.biomedcentral.com/articles/10.1186/1471-2288-14-75
(4) James et al., Introduction to Statistical Learning
(5) https://dinsdalelab.sdsu.edu/metag.stats/code/randomforest.html
(6) http://www.doc.ic.ac.uk/~sgc/teaching/pre2012/v231/lecture13.html

Software: https://www.r-project.org/ https://www.rstudio.com https://cran.r-project.org/web/packages/caret/index.html https://cran.r-project.org/web/packages/mice/mice.pdf https://cran.r-project.org/web/packages/randomForest/randomForest.pdf https://cran.r-project.org/web/packages/kernlab/kernlab.pdf https://cran.r-project.org/web/packages/RSNNS/RSNNS.pdf https://cran.r-project.org/web/packages/VIM/index.html