Brian Kim
CPE - 646
Final Project

Predicting Adult Onset Diabetes Using Machine Learning

Table of Contents

1. Project Overview

1.1 Problem Overview

According to the Center of Disease Control, 9.8% of all Americans, or 30.3 million people, were estimated to have diabetes. Likewise, in that same year, the CDC estimated that 33.9% or 84.1 million Americans had prediabetes, a condition in which blood sugar levels are higher than normal but not enough to be considered diabetic.[1] CDC statistical studies from 1994 to 2015 shows a large trend towards increasing age-adjusted prevalence of both diabetes and obesity rates in the adult population in the U.S.[2] In 1994, only a single U.S. state had a percentage of adults diagnosed with diabetes in a range of 6.4 - 7.4% or greater. Similarly, only two U.S. states had adult obesity rates of 18 - 21.9% or greater that year. By 2015, all U.S. states had diagnosis rates of 6.4 - 7.4% or greater with 32 states having 9% or greater. The obesity rates showed a similar trend with all but 1 state having obesity rates greater than 22%, and 36 states at 26+%. Early intervention for the over 80 million Americans at risk for developing adult onset diabetes is necessary in order to alleviate this ever increasing issue. Identifying individuals at risk could provide relief for the healthcare system by assisting medical professionals with taking early preventative measures before conditions worsen in high risk individuals.

1.2 Data Overview

In order to reduce the complexity of studying type 2 diabetes, researchers sought to focus on a population with low genetic diversity. It was discovered that the Pima Indians of Arizona had the highest percentage of type 2 diabetes in the world. Interestingly, the Pima indians also had almost no instances of type 1 diabetes among its younger population.[3] The low genetic diversity and high rates of only adult onset diabetes makes this population an excellent control group to study this disease. The specific data set contains medical predictor variables. These diagnostic measures are used by medical professionals when diagnosing patients with type 2 diabetes. The different variables are as follows: number of pregnancies, plasma glucose concentration after two hours in an oral glucose tolerance test (mg/dl), blood pressure (mmHg), triceps skin fold thickness (mm), 2-hour serum insulin (μIU/mL), BMI, diabetes pedigree function, and age. The data was split into two segments. 90% would be used in training and 10% would be used for testing. The validation of the data set for the Neural net will utilize the entire test set.

The algorithms in this project will utilize data that is first normalized using min-max normalization using the norm_mean.m function. Results from using the normalized data versus data transformed using PCA will be compared and analyzed.

2. Initial Analysis of Data Set

2.1 Histogram Plot Analysis

Several observations can be made from the histogram plots for each feature split by positive and negative diagnosis. From the glucose plot (figure 2), it can be observed that for negative cases, the distribution is relatively normal, centered at 100 mg/dl. For positive cases on the other hand, the samples tend towards the larger values. Based on this graph, glucose may be an important feature in determining a positive diabetes diagnosis. Blood Pressure on the other hand (figure 3), may not be as useful of a metric. The distributions for both positive and negative cases tends to center at around 70-80 mmHg. The larger sample size for negative diagnosis contributes to the relative size difference of the distribution however. The plots were created using the histPlot.m function. This function plots two histograms, one for each class, on a single plot for every feature.

2.2 Handling Missing Data

Another interesting observation from these graphs is the prevalence of missing data. In both the skin thickness (figure 4) and insulin (figure 5) plots, there is a large number of values in the 0 bin. Though it would be possible to have skin thickness values in the 0-10 range, a value of 0 would not be possible. A value this low would also not be possible for insulin. Furthermore, after viewing the values for these features, it was verified that indeed there were significant missing values. Several approaches were considered for fixing this issue. The first option was to drop samples with missing data. This option was not chosen as doing so would lead to a loss in over 30% of all sample data. The second option was to drop the features with significant missing values. Since the missing values were mainly isolated to the skin thickness and insulin features, it would be possible to simply remove these features instead of dropping the entire sample. The final consideration was to estimate the missing values by replacing zero values with the mean of the existing values. This approach, while it could skew results if positive values are sufficiently large enough to pull the mean towards it or vice versa, would still allow the feature to be utilized. Ultimately, both options two and three were chosen. There would be two tests, one with the full data set including estimated

values, and another with the “bad” features removed. The missing data issue was resolved by creating the estMissFeature.m function. This function estimates the missing features by replacing 0 values in “bad” features with the mean of the existing values. The user must give a vector indicating which features are “good” and which are “bad”. For example, the number of pregnancies is a feature that can be 0, whereas BMI cannot. Therefore the latter would be 1 in the error vector whereas error vector value for the former would be 0.

2.3 Scatter Matrix - Feature Covariance

Finally, a scatter matrix (figure 10) was created to see if there was any correlation between features. From this plot, it was hypothesized that there is a moderate positive linear relationship between skin-fold thickness (ST) and BMI. Likewise, it was hypothesized that there is a weak positive linear relationship between the glucose (G) values and the insulin (I) values. In the case of skin-fold thickness and BMI, it is reasonable to assume these are correlated based on outside knowledge. Generally, high BMI corresponds with high body fat percentages, which would then correlate with higher skin-fold thickness values. Since both ST and I were considered “bad” features and they seem to correlate with “good” features to some extent, the initial hypothesis was that the removal of features may not negatively impact performance as much as initially presumed. The scatter matrix was plotted using the function scatterMatr.m. This function was designed to take in a data set in which the features are represented by columns and a title string, and create a scatter matrix to determine correlation between the features.

2.4 Dimension Reduction - PCA

Dimensionality is a major factor that influences computational requirements and system complexity. Though the data set only has 8 features, which is not a significant amount that would warrant dimension reduction, the decision to use PCA or principal component analysis was made to improve the understanding of the data set and to determine which classification algorithms may be best suited for this data set since PCA could be used to visually observe the nature of the data set in three dimensions. From figure 11, it can be seen that the data set for both positive and negative cases are fairly clustered together. There is significant overlap and the data appears to not be linearly separable at least in three dimensions. Several trials will be taken testing different numbers of principal components and its effects on the results. Likewise, results from the data set transformed using PCA and not will be compared. The functions pcProj.m and princComp.m were created to project a given data set using PCA. The princComp

function returns the requested number of principal components whereas the pcProj function projects a dataset using the principal components. Results were gathered using 3 and 4 principal components. 3 principal components were responsible for 68.38% of total variance whereas 4 PC were responsible for 78.58% of variance. The results comparison table can be viewed in section 4.

3. Classification Methods

3.1 kNN - k - Nearest Neighbors

The first classification method used was the k-Nearest Neighbors. Though based on initial observation of the PCA transformed data confirmed that it would not be linearly separable, there were still fairly distinct clusters for each class. The outliers were for the most part significantly surrounded by a majority of the opposing class. The k-NN algorithm was therefore considered as a viable option. k-NN was implemented using the twoC_knn.m function. This function takes as input the training data set and the test data set. For every sample in the test data set, it finds the k nearest neighbors in the training set. The k is set to ensure that it is rounded up to the nearest odd integer. The relationship between k and the number of samples can be seen below.

```
29        %k is equal to the square root of n, rounded up to the nearest odd
30        %integer.
31 -      k = 2*floor(sqrt(n)/2)+1;
```

Determining k number for nearest neighbors

The nearest neighbor algorithm was implemented using the euclidean distances between two points. The euclidean distance is found using the following equation:

```
%Get the Euclidean distance between test sample m and training
%set samples.
E = sqrt(sum(((tDnC - teD(i,:)).^2).').');
```

Euclidean distance between test sample and point in training set

The class labels of the k - nearest neighbors are then counted and compared. The test sample is then set to the class corresponding to the greater value.

3.2 SVM - Support Vector Machine

The next model used was the support vector machine. The support vector machine is a model in which a data set is separated with one or more hyperplanes. The implementation of the support vector machine was used using Matlab's SVM functions within the Statistics and Machine Learning toolbox. The decision to use Matlab's functions was made due to the significant margin of error when attempting to find the best separating hyperplane. Due to the inability to linearly separate the data set and the relatively close clumping of the data set, the results from a self made SVM model were quite poor.

Three different approaches using SVM were compared. The first was to use the full data set normalized. The second approach was to use the full normalized data set with a gaussian kernel function. Lastly, the final approach was to use the PCA transformed data set.

Training an SVM model in Matlab is fairly simple. The following code was used to create the SVM models.

```
92      %% SVM
93      %Train SVM Model
94 -    SVMModel = fitcsvm(TrNorm.',TrClass);
95
96      %Train SVM Model with PCA data.
97 -    PCA_SVM_Model = fitcsvm(projData(:,(1:size(PC,2))),TrClass);
98
99      %SVM gaussian function kernel
100 -   SVMKernMod = fitcsvm(trSetEst(:,1:(size(trSetEst,2)-1)),trSetEst(:,size(trSetEst,2)),...
101     'Standardize',true,'KernelFunction','gaussian','KernelScale','auto');
```

Training SVM models in Matlab

These SVM models can then be used to classify unknown samples using the predict method. The output from this method is the label and score, however for the purposes of this project, only the label was used to determine overall accuracy on the test set.

3.3 Neural Net

The final classification method chosen was the neural net. The neural net chosen functions using a feedforward and back-propagation algorithm. The training protocol used was online training of 100 epochs. At the moment thresholding is present but unsupported due to an optimization issue when attempting to find the best threshold limit. Since the values are relatively small, future testing can be done to determine an

optimal threshold, however due to the relatively small size of the data set, using a set number of epochs is not a demanding task for a CPU to run.

To create a neural network, a class was developed in Matlab, called NN.m. This NN class has 4 properties, or private data values. These properties can be seen below:

```
6       properties
7           w1 %weights for input -> hidden layer
8           w2 %weights for hidden layer -> input
9           b1 %biases for hidden layer neurons
10          b2 %bias for output layer
11          %Jw %current J(w), training error.
12          %thrReached %1 if Threshold theta was reached.
13      end
14
```

Neural Network class NN properties

The constructor for this class was initialized in the following manner.

```
16      function nn = NN(i,j)
17          % NN Construct an instance of a neural net.
18          %   input:
19          %   i: number of features in data set.
20          %   j: number of hidden layers.
21          %   Initializing a NN obj for the first time will create
22          %   a neural net object with random weights and biases between
23          %   -2 to 4.
24          %   Number of hidden layers is equal to the mean of the sum of
25          %   the number of input and output layers.
26          %
27          %% *Developers note
28          %   Jw and threshold theta is currently unsupported due to
29          %   problems with determining the optimal threshold.
30          %% Set Initial Properties. Random Weights and zero bias.
31          nn.w1 = 4*rand(j,i)-2;
32          nn.w2 = 4*rand(1,j)-2;      %Single output neuron.
33          %bias initialization.
34          nn.b1 = zeros(j,1);
35          nn.b2 = 0;
36          %nn.Jw = 0;
37          %nn.thrReached = 0;
38      end
```

Neural Network NN constructor

The constructor inputs i and j are the number of features in the input and the number of hidden layer neurons respectively. The weight matrices from input to hidden

and hidden to output are randomized in the range [-2 4]. Since the classification problem is binary, this neural net class currently only supports a 1 neuron output layer.

The two methods for this class are train and classify. The train method utilizes a feed forward and back propagation algorithm. The feedforward portion takes an input value, calculates the result as it goes through the neural network, and returns a predicted label for the sample. The back-propagation algorithm is responsible for the training. The back-propagation algorithm was implemented using gradient descent. The true tag value is compared with the value at the output node. These two methods were handled in the NN.train method. The classify method takes in an unlabeled data set and returns a column vector with the predicted label. Since the activation function used is the sigmoid, if a value at the output is greater than 0.5, it is predicted to be a 1 or positive diagnosis, else it is predicted to be a 0, or negative diagnosis.

In the train method, the following equations were implemented to change the weights of the network connections.

$$\Delta w_{kj} = \eta(t_k - z_k) f'(net_k) y_j$$

Modify weights from hidden to output layer

$$\Delta w_{ji} = \eta \left(\sum_{k=1}^{c} w_{kj} \delta_k \right) f'(net_j) x_i$$

Modify weights from input to hidden layer

The biases were modified based by adding the difference between the true label and the output, weighted with the training rate η or “r” in the code.

4. Results and Analysis

4.1 Performance Table for Full Data set

Performance on Full Data Set	
PCA	
kNN - 3 PC	0.7500
kNN - 4 PC	0.7985
SVM - 3 PC	0.7895
SVM - 4 PC	0.8026
Standardized	
SVM	0.7895
SVM Kernel	0.7500
Neural Net - 8	0.7895
Neural Net Fit - 8	0.7789
Neural Net - 6	0.7763
Neural Net Fit - 6	0.7673
Neural Net - 4	0.7763
Neural Net Fit - 4	0.7760

The performance was calculated based on how accurately the model was able to correctly classify the 10% of the original data set reserved. These samples were not used in the training portion of the models. The performance is the ratio of correctly labeled samples divided by the total number of samples in the test data set.

The best performing model was the SVM using 4 principal components. This is not surprising since the 4 principal components account for 78.58% of total variance while utilizing half the dimensions of the full data set.

The performance of the kNN algorithm is reasonable since there were significant “outliers” of one class in the other as seen in figure 11. The pcPlot3D.m function can be used to visually confirm the significant amount of outliers that clearly contributed to the error using k Nearest Neighbors.

The performance of the neural net slightly increases as the number of hidden layers is increased. However, this increase is fairly marginal, and there is no increase from 4 to 6. Likewise, the Fit performance, which is the Neural Net attempting to classify its training set, decreases from 4 to 6 hidden neurons, then slightly increases at 8 hidden neurons. The

performance of the Neural Net can change depending on the initial randomized values, however, running the Project.m script several times can result in a performance value similar to the table above.

4.2 Performance Table for reduced Data set

Performance on Reduced Data Set	
PCA	
kNN - 3 PC	0.7500
kNN - 4 PC	0.7763
SVM - 3 PC	0.8026
SVM - 4 PC	0.7895
Standardized	
SVM	0.8026
SVM Kernel	0.8026
Neural Net - 8	0.7763
Neural Net Fit - 8	0.7717
Neural Net - 6	0.7895
Neural Net Fit - 6	0.7775
Neural Net - 4	0.7763
Neural Net Fit - 4	0.7731

The table above shows the performance when the skinfold thickness and insulin features are removed. In this scenario, SVM is again the winner, with equal performance when using 3 principal components and when using the standardized data set with or without a gaussian kernel function. Since these features were those with a significant number of missing values, the estimation using the existing mean was most likely responsible for the outliers present in the data set. This reduction in outliers most definitely contributed to greater accuracy when linearly separating with hyperplanes.

kNN had a slight dip when using 4 PC. This could possibly be explained by the same situation as with the SVM. However, in the case of kNN, a reduction in the number of outliers will reduce the possibility of correct classification as now there are less outliers within the same region dominated by the opposing class.

The Neural Net had a little change, however, the change in performance when decreasing the input feature space did not correlate with the number of hidden neurons.

Change in Performance

Change In Performance	
PCA	
kNN - 3 PC	0.00%
kNN - 4 PC	-2.78%
SVM - 3 PC	1.66%
SVM - 4 PC	-1.63%
Standardized	
SVM	1.66%
SVM Kernel	7.01%
Neural Net - 8	-1.67%
Neural Net Fit - 8	-0.92%
Neural Net - 6	1.70%
Neural Net Fit - 6	1.33%
Neural Net - 4	0.00%
Neural Net Fit - 4	-0.37%

Above is the change in performance when the feature space is reduced. The most significant improvement is the SVM Kernel at 7% improvement. The removal of “bad” features most certainly helped improve the linear separability of the classes as there were less outliers within the opposing class due to the mean estimation of missing features. The greatest decrease in performance however was the kNN using 4 principal components. A possible explanation was posited in the previous section. Since a large number of outlier samples were clustered in the opposing region, these would have contributed to the number of similar neighbors when labeling an unknown. However, removing many of these outliers would reduce the positive effect of clustered outliers in labeling the unknown when using k nearest neighbors.

5. Conclusion

5.1 Future Recommendations

There are several potential avenues of improvement for this project. The most obvious method to improve overall performance would be to collect more data samples. A larger training data set that accurately represents the true distribution will definitely have a net positive impact on performance for all algorithms.

However, a potential method for improvement could be to use a different activation function in the neural network. In the Unused functions folder within the functions folder, there are functions to implement both the Rectified Linear Unit and a logistic sigmoid function. In the current build, these functions were left out as there were issues in implementing a string/label input into a matlab function. Normally, as in the case with the SVM built in matlab function, if the user does not input the string, there is a default value used. However, there is an error in calling the function since the method call would be missing a value. A quick-fix solution could be to use a switch or case statement and provide a key mapping for the user, however multiple case statements would be required in this scenario whenever an activation function is used. Another solution could be to implement separate methods for each activation function, though this would also be redundant and inefficient.

6. References

[1] https://www.niddk.nih.gov/health-information/health-statistics/diabetes-statistics

[2] https://www.cdc.gov/diabetes/statistics/slides/maps_diabetesobesity_trends.pdf

[3] http://diabetes.diabetesjournals.org/content/53/5/1181

7. Figures

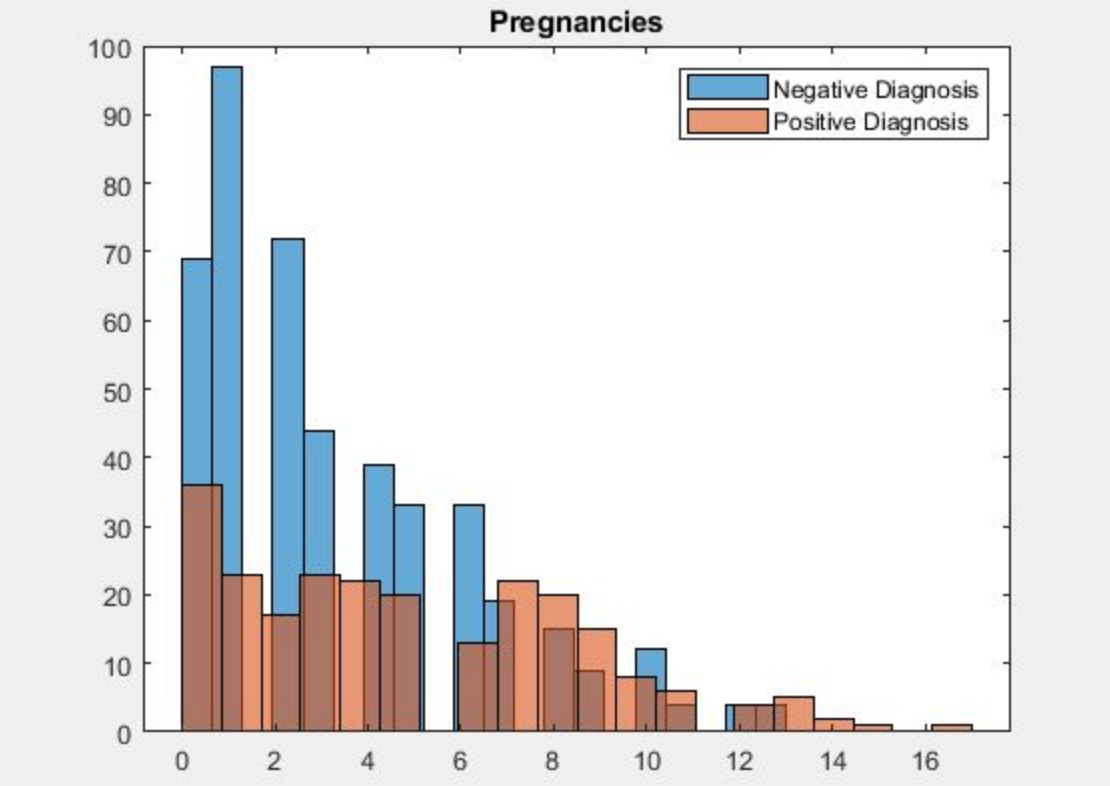

Figure 1

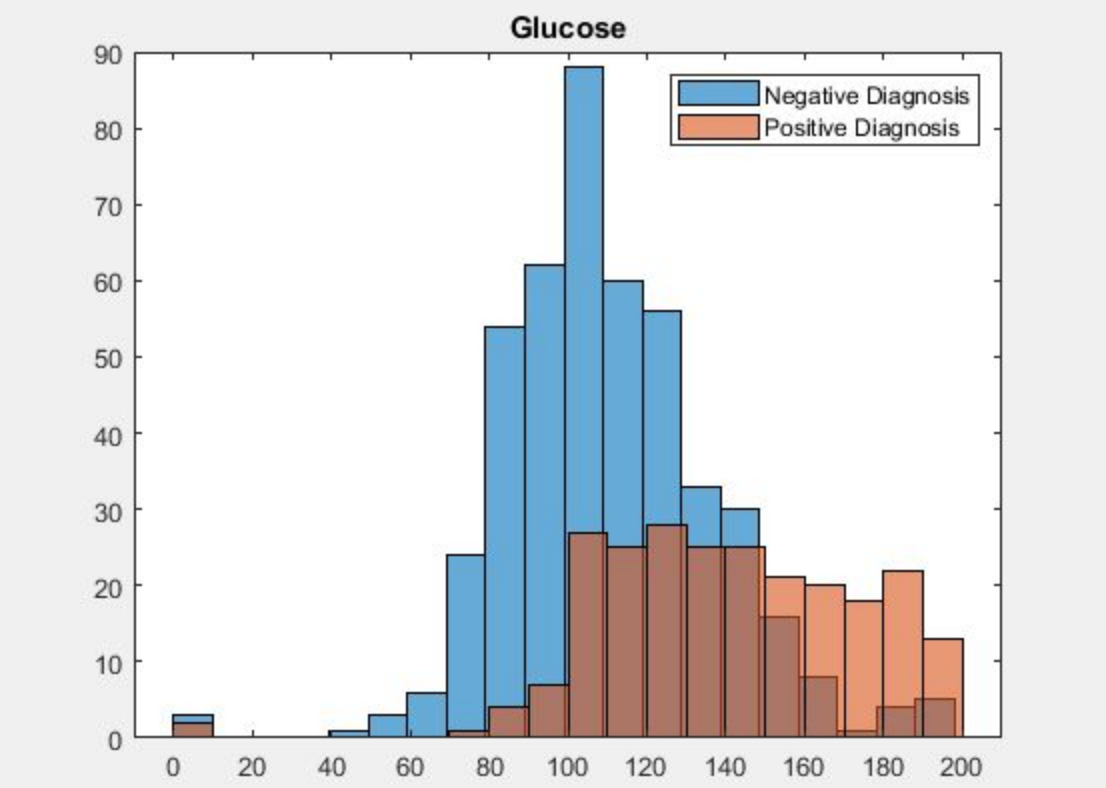

Figure 2

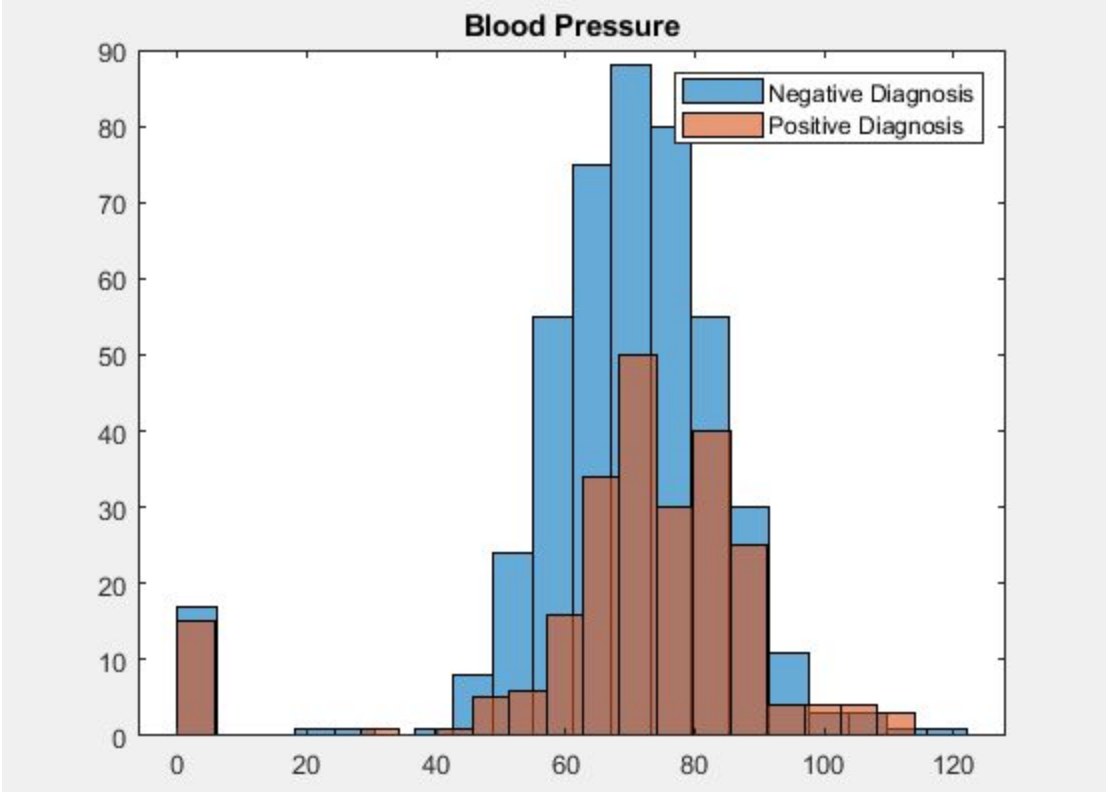

Figure 3

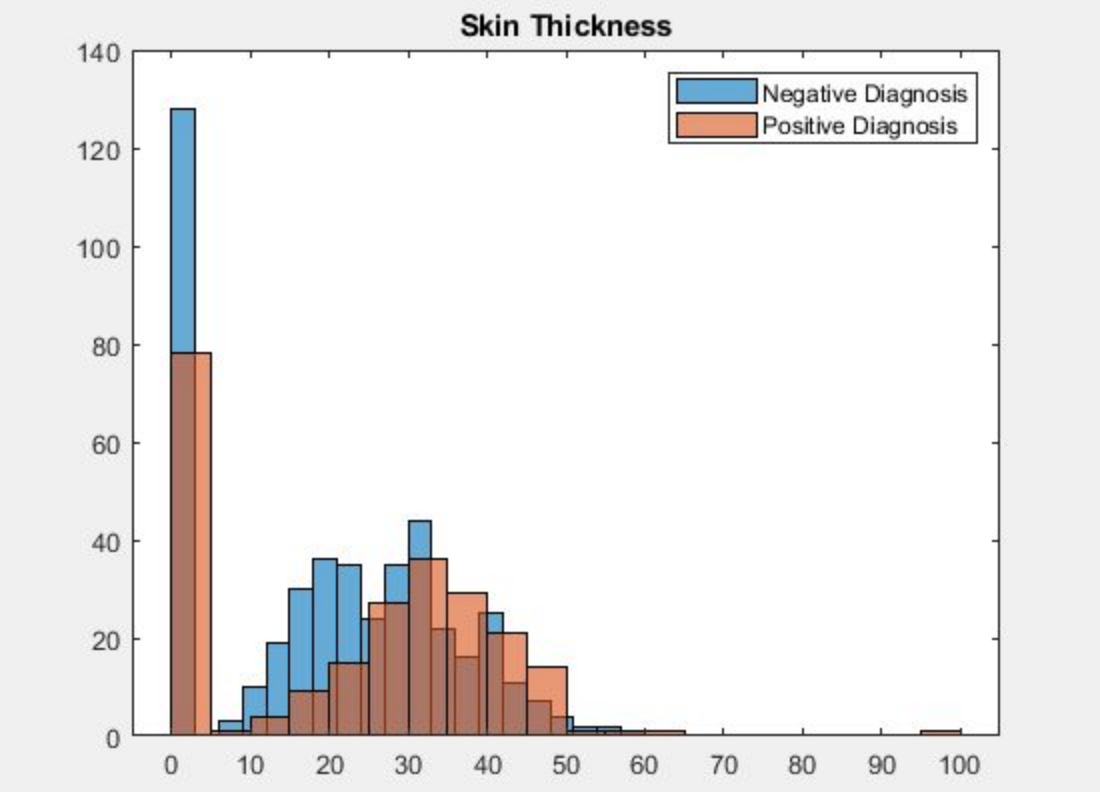

Figure 4

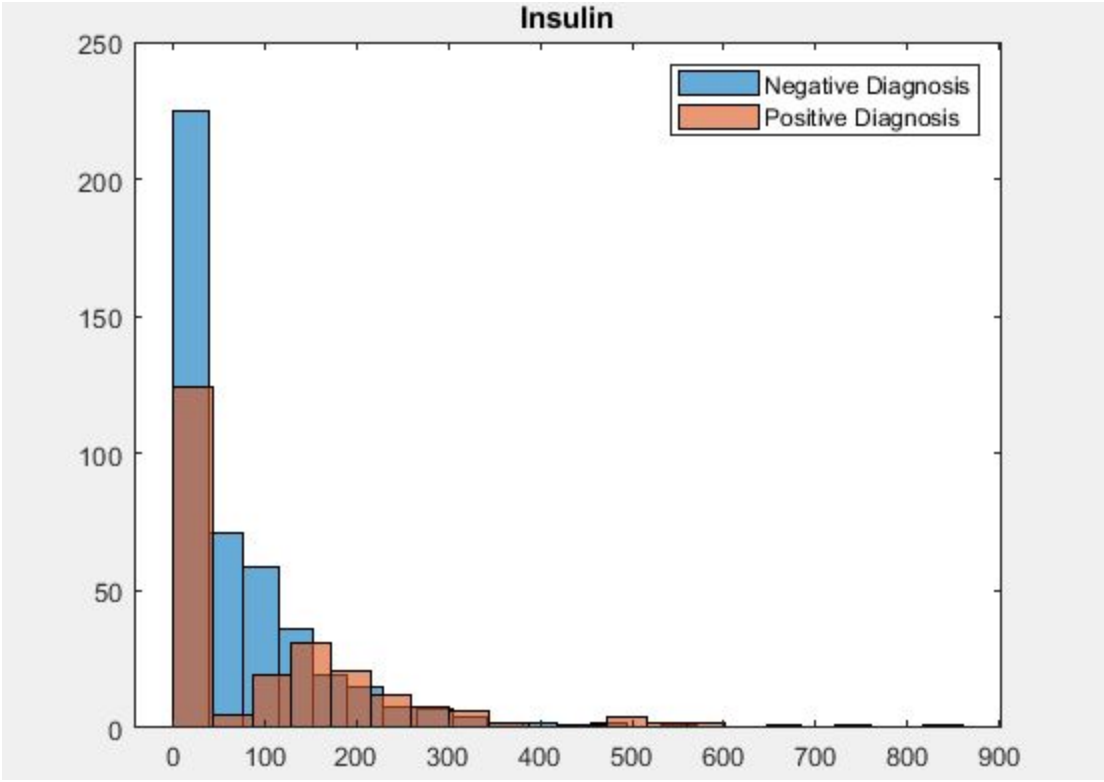

Figure 5

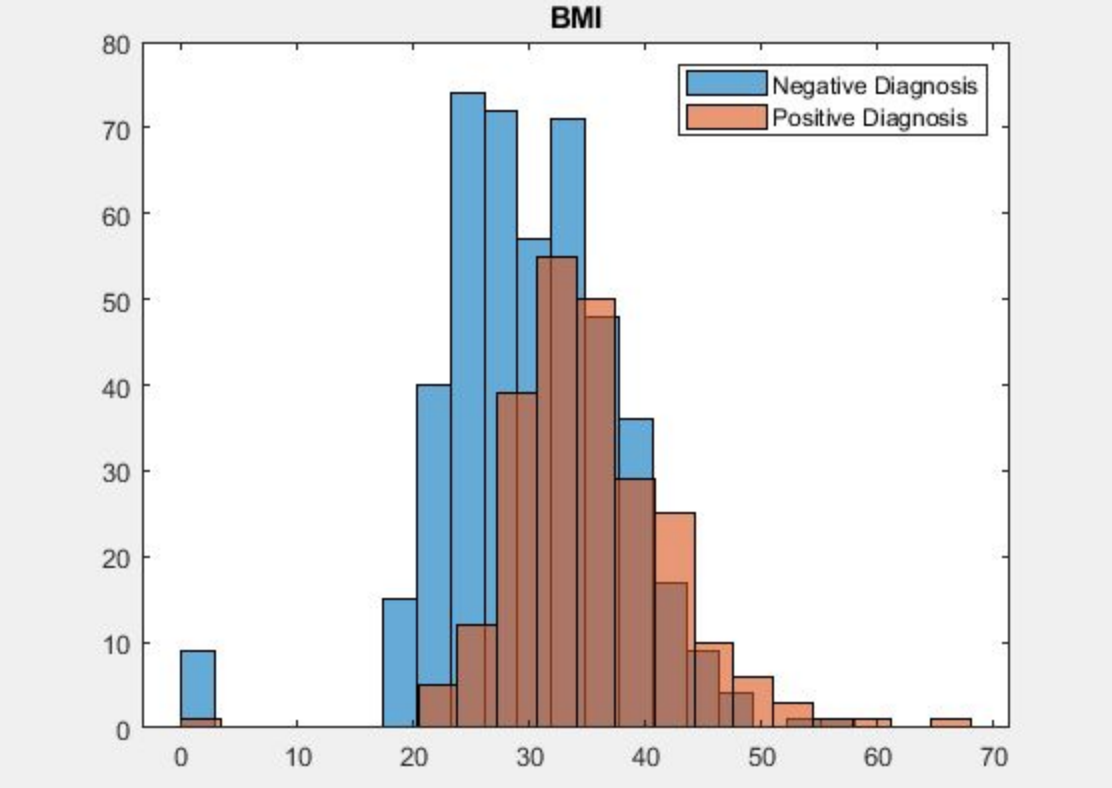

Figure 6

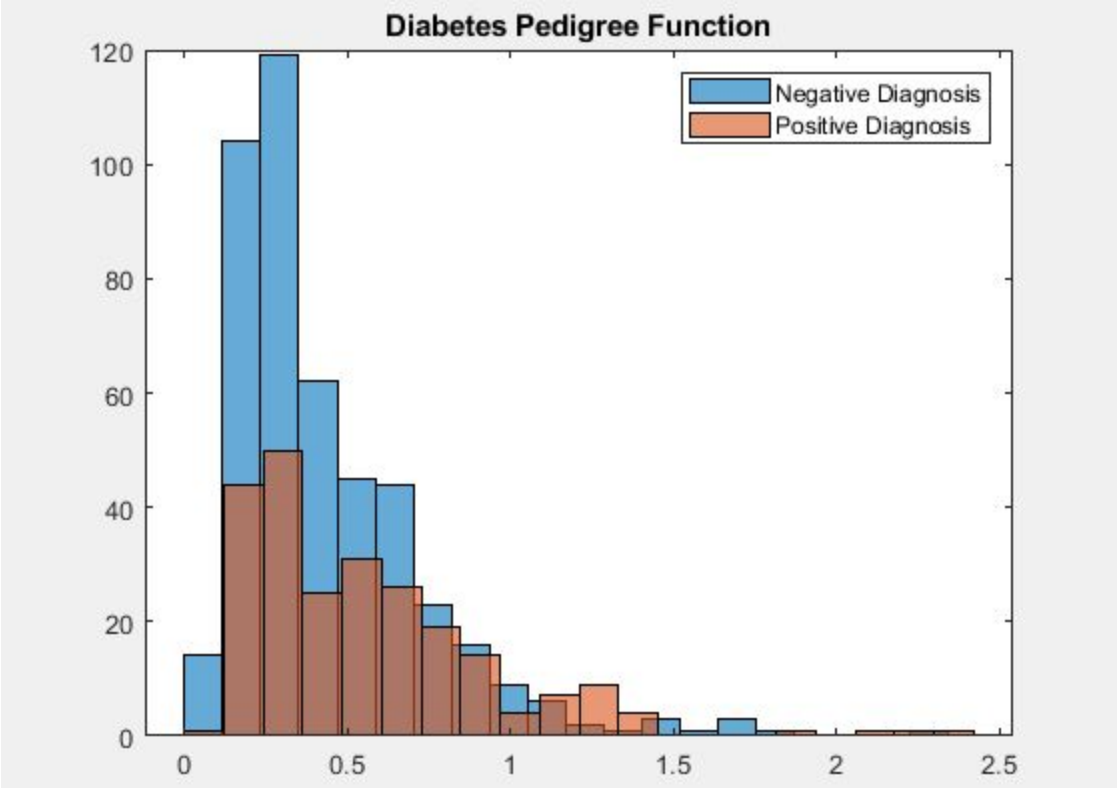

Figure 7

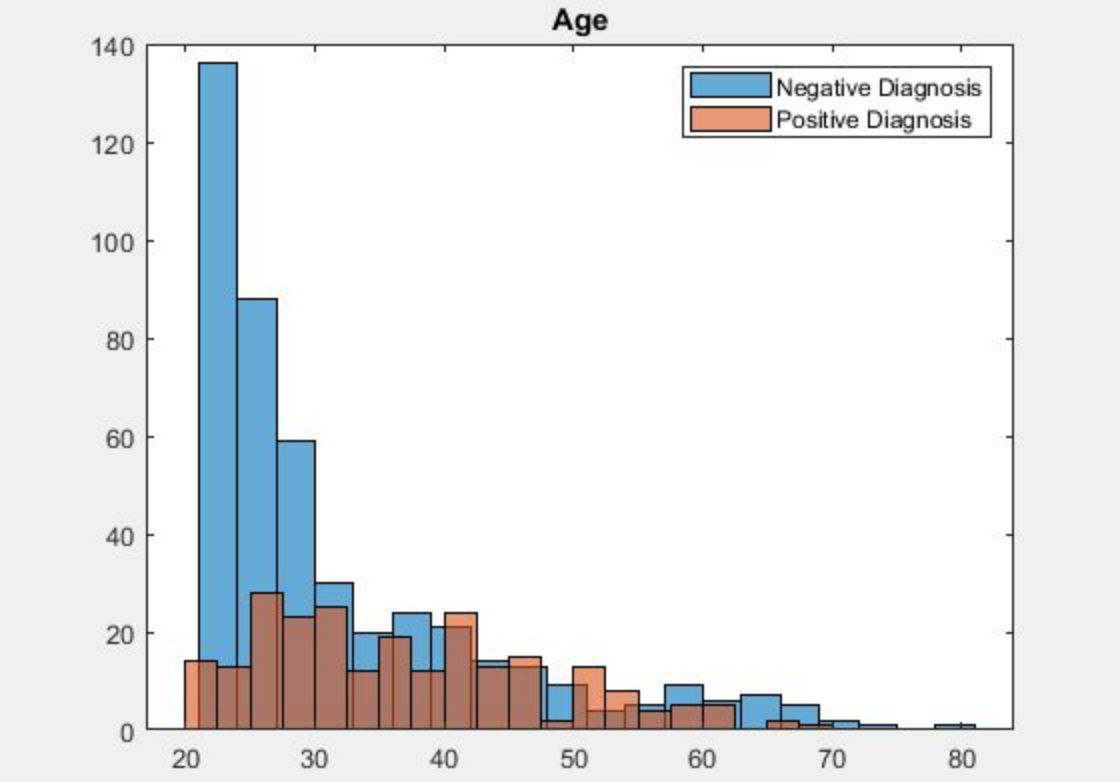

Figure 8

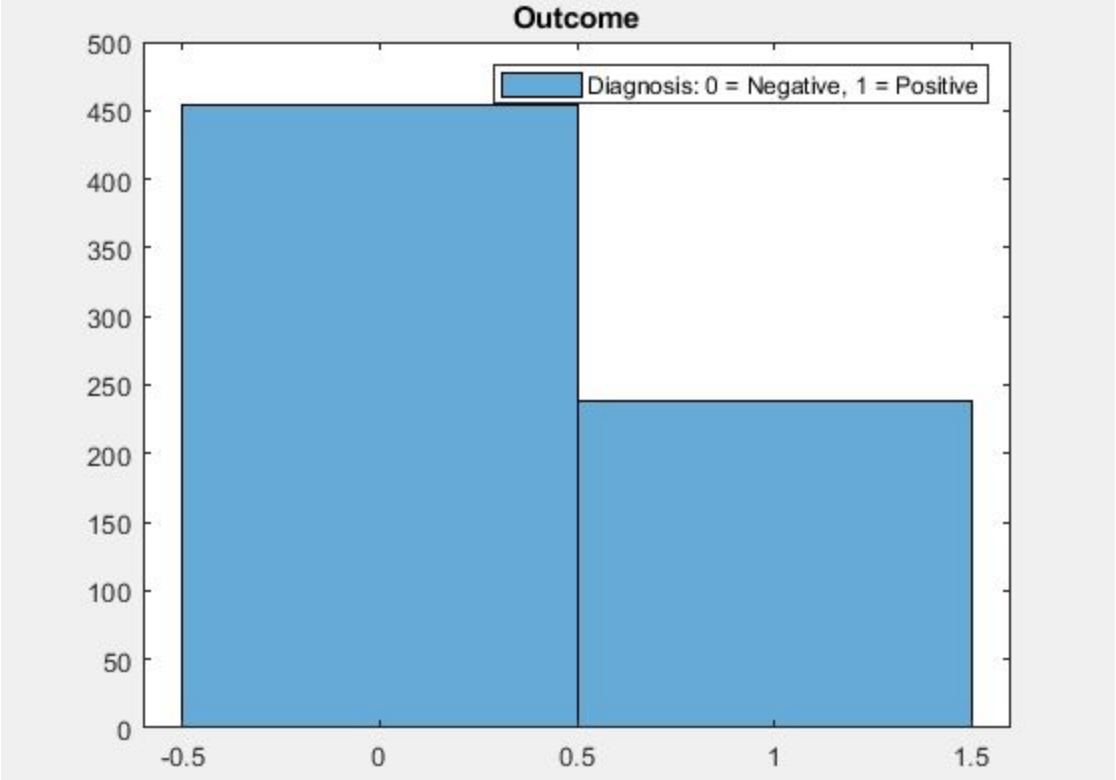

Figure 9

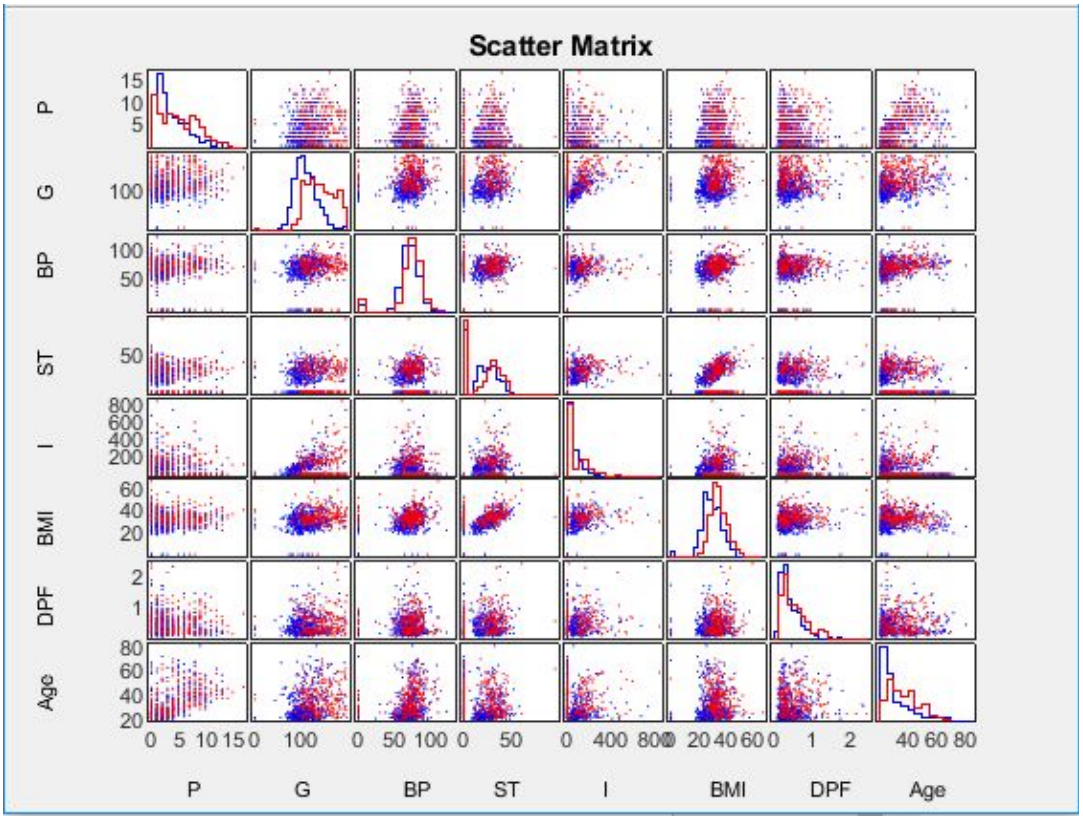

Figure 10

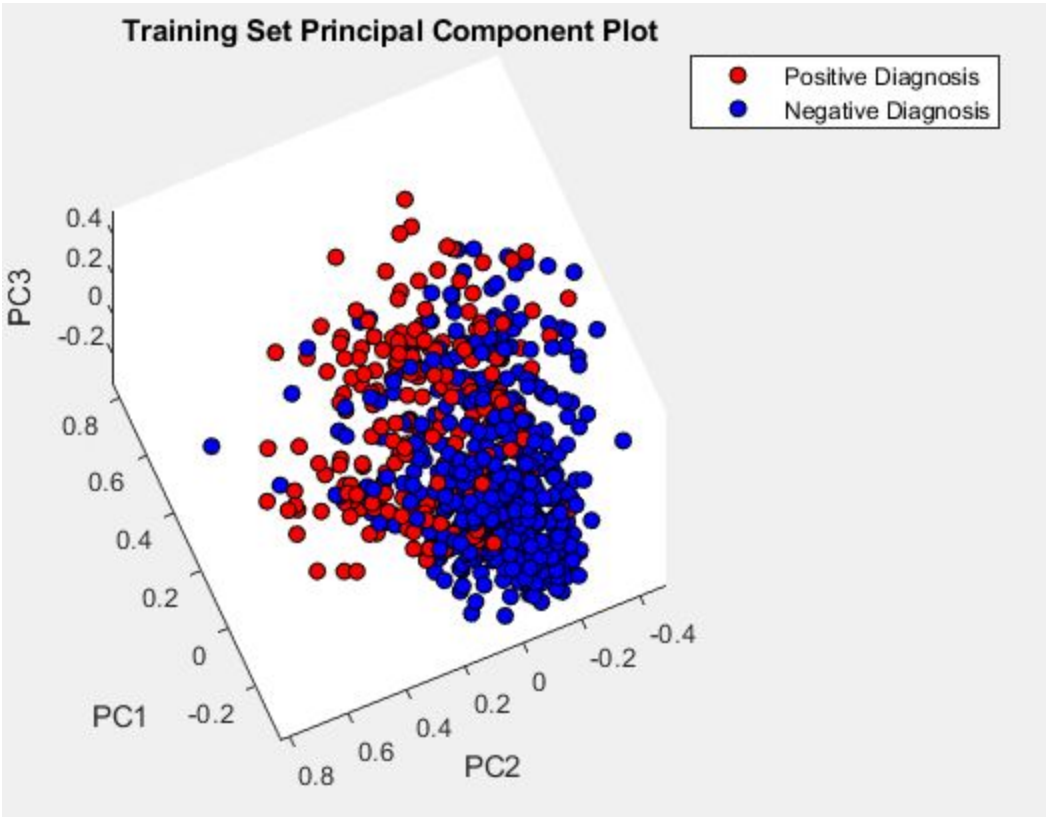

Figure 11