Lab 1 - Machine Learning

Daniel Persson (danpe586), Jaskirat Marar (jasma356),
Hussnain Khalid (huskh803)

25/1/2022

Statement of Contribution

For solving this lab, the group decided to split the responsibility equally by assigning 1 question to each member. The split by mutual consensus was as follows:

1. Assignment1: Solution and report by Daniel Persson
2. Assignment2: Solution and report by Jaskirat Marar
3. Assignment3: Solution and report by Hussnain Khalid

We were able to communicate with each other effectively and responsibly. All the group members were forthcoming in discussing issues being faced while solving the problems. We were able to each present our solution to the others well before the deadline and were able to conclude on the structure and content of the final report.

"*We acknowledge that each member has contributed fairly and equally in solving this lab.*"

By undersigned:

Daniel Persson

Jaskirat Marar

Hussnain Khalid

Assignment 1. Handwritten digit recognition

The confusion table for the train data:

```
##     prediction
## true   0   1   2   3   4   5   6   7   8   9
##    0 177   0   0   0   1   0   0   0   0   0
##    1   0 173  10   0   0   0   1   0   1   3
##    2   0   0 169   0   0   0   0   2   2   0
##    3   0   0   0 196   0   2   0   1   0   1
##    4   0   2   0   0 167   0   2   6   2   0
##    5   0   0   0   0   0 183   1   2   0  11
##    6   0   0   0   0   0   0 200   0   0   0
##    7   0   1   0   1   0   1   0 192   0   0
##    8   0  14   0   1   0   0   2   0 188   0
##    9   0   4   0   3   2   0   0   2   4 181
```

The confusion table for the test data:

```
##     prediction
## true  0  1  2  3  4  5  6  7  8  9
##    0 97  0  0  0  0  0  1  0  0  0
##    1  0 91  3  0  0  0  0  0  0  3
##    2  0  0 93  1  0  0  0  0  1  0
##    3  0  0  0 94  0  0  0  2  1  1
##    4  1  0  0  0 89  0  1  5  1  3
##    5  0  1  0  1  0 79  0  0  0  6
##    6  0  0  0  0  0  0 94  0  0  0
##    7  0  3  0  0  0  1  0 90  1  0
##    8  0  3  0  1  0  0  1  0 86  0
##    9  0  2  0  3  0  0  0  2  1 93
```

It can be seen that zeroes, twos and sixes are easiest to predict with only a few misclassifications in each data set. The digit that has the highest misclassification in the training data is eight which is classified as a one. In the test data it is digit four that has most misclassifications. It can be seen that misclassified fives often are classified as a 9, and four misclassified as a 7.

The misclassification errors for the train and test data are:

```
## [1] 85
```

```
## [1] 50
```

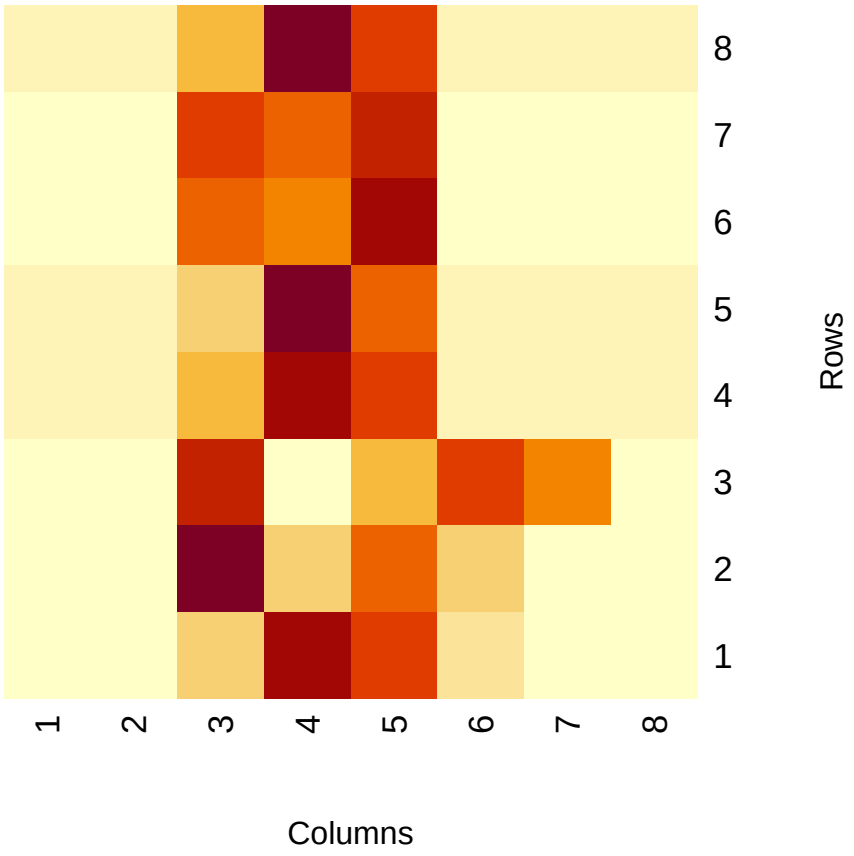
respectively. The test data has more errors relative to its size than the train data. A reason can be that the train data was used for training while test data is new data.

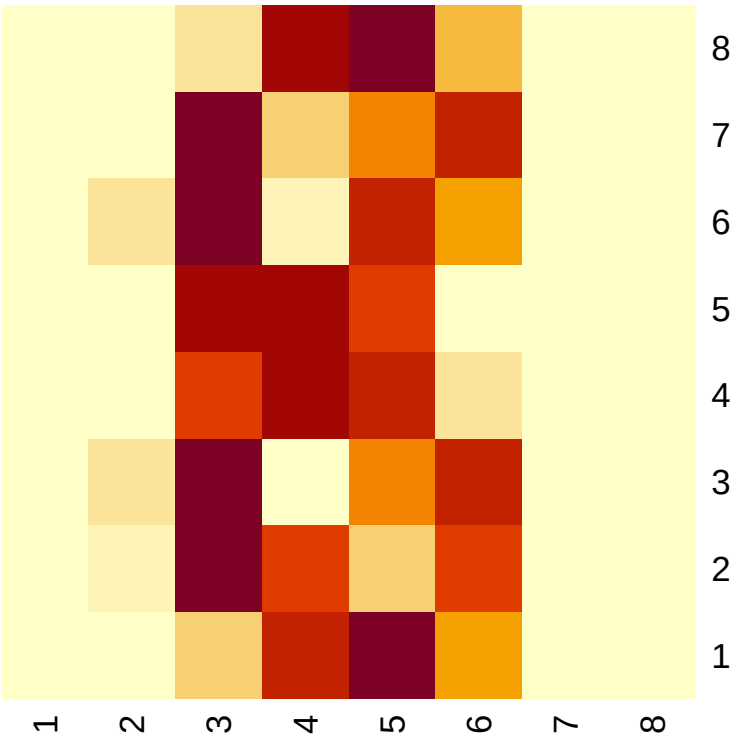
The overall prediction accuracy for train and test data:

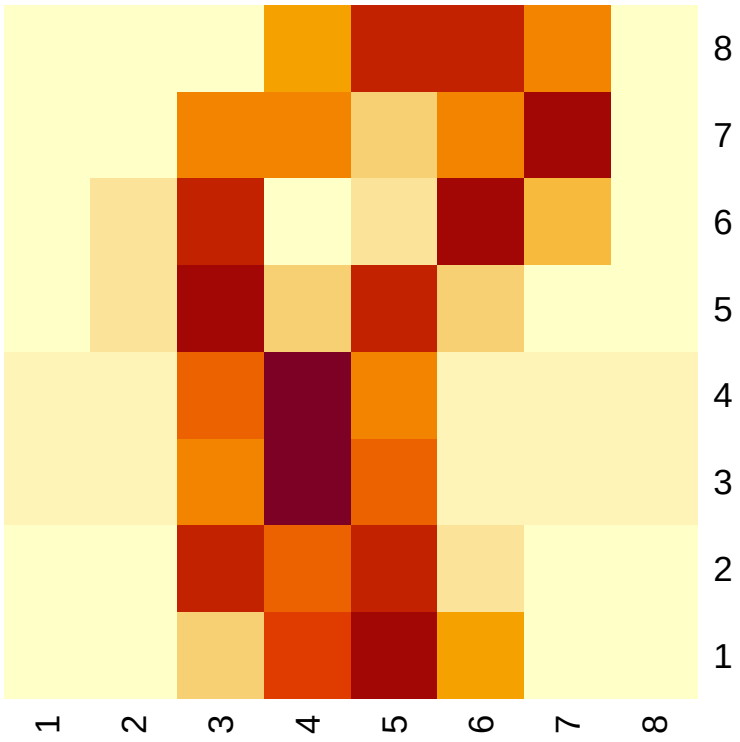
```
## [1] 0.9555207
```

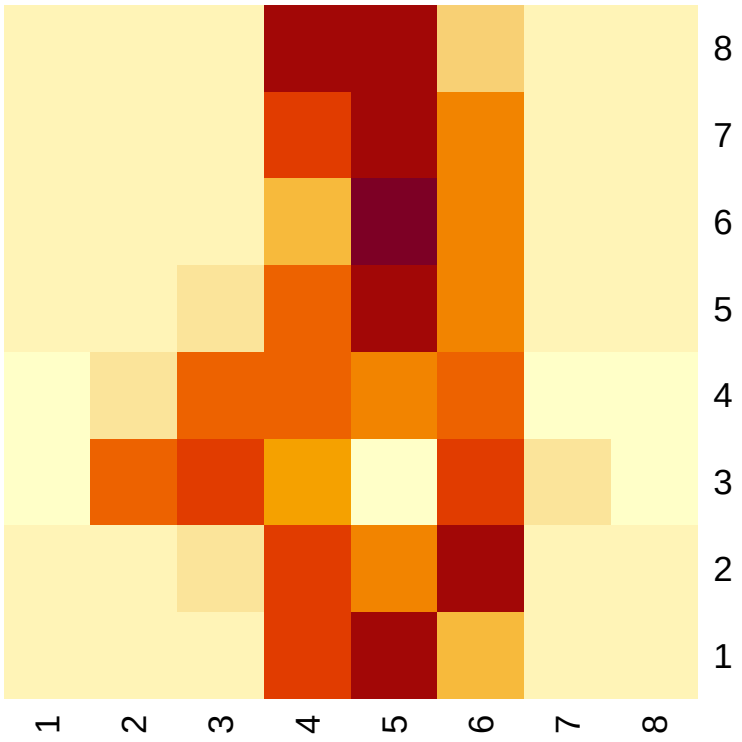
```
## [1] 0.9476987
```

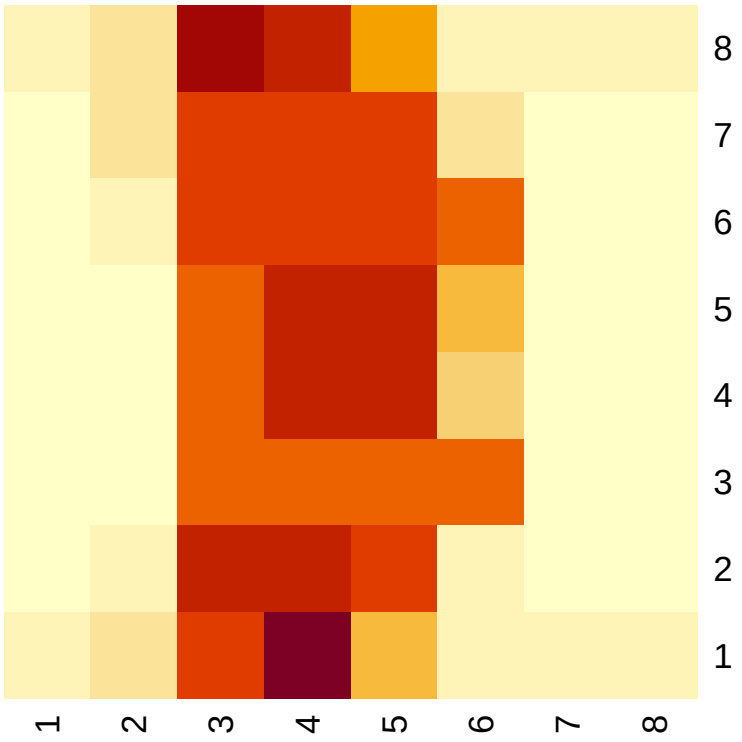
respectively. The accuracy are quite similar for the tested train and test data but as stated above the accuracy is higher for the train data.


8
7
6
5
4
3
2
1
Rows
1
2
3
4
5
6
7
8
Columns

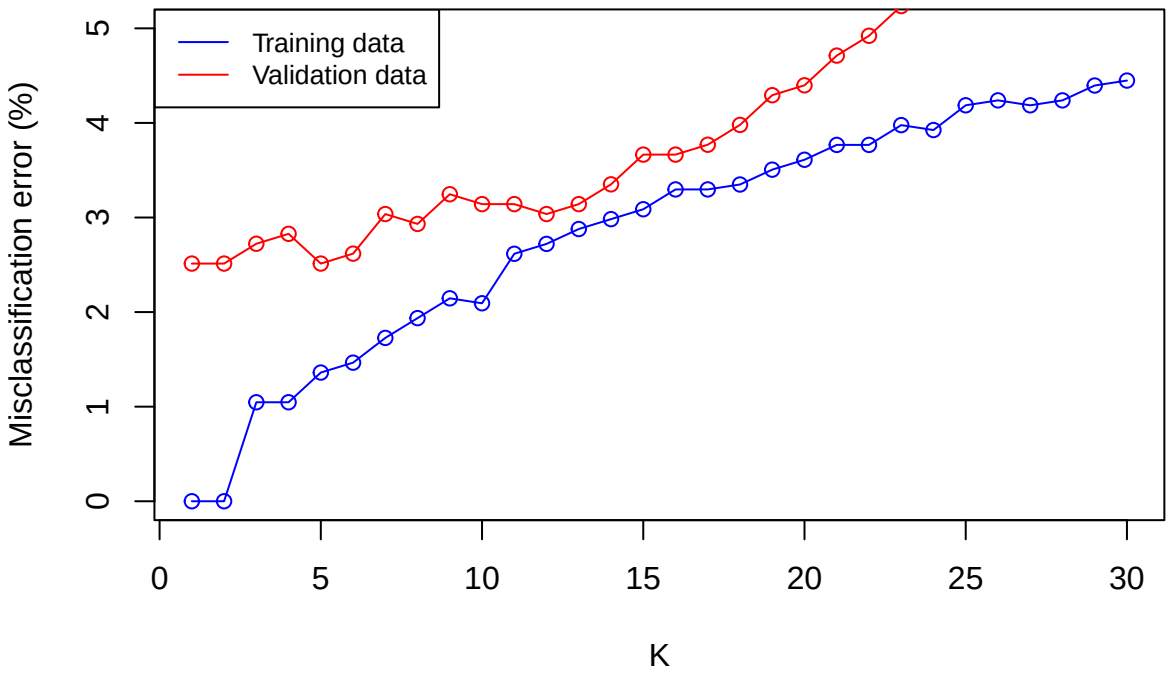

8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8


8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8


8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8



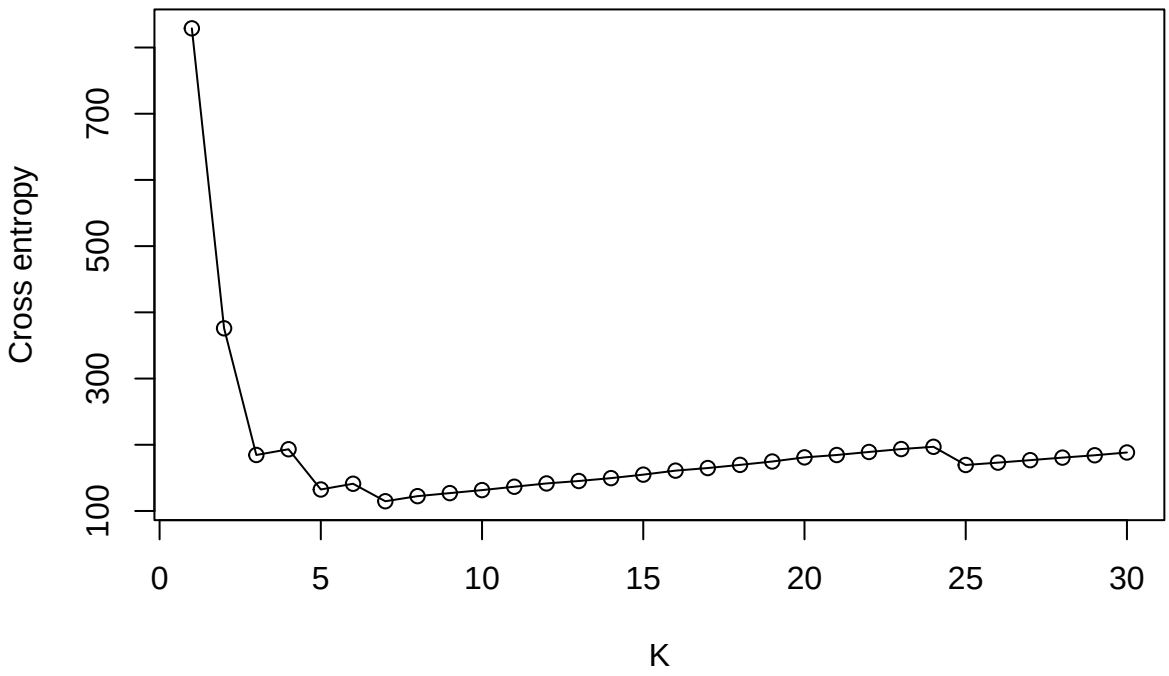
The two first eights that were easiest to classify has the shape of an eight and should be easy. The three last eights was hardest to classify and the shape of the 2 last does actually not look like an 8.



The model complexity decreases as K increases, and the overall training error increases. $K = 5$ is chosen as the optimal K, since the validation errors decrease to a local minimum for $K = 5$ and the error is about the same as for $K = 1, 2$. $K = 5$ is also less complex than the lower K's.

Finally, the test error for $K = 5$ is calculated to:

```
## [1] 0.03138075
```



According to the graph for the cross entropy the optimal K should be 7, which is different from the misclassification error. The cross entropy might be a better choice for this assignment since it can take more than one distribution into account. Using cross-entropy makes more sense because this is multiclass classification problem and hence our loss function needs to take into account multiple probability distributions. Therefore, in our problem, by using cross-entropy we are essentially minimizing the distance between the probability of the predicted value and the known value from the data. This is why using cross entropy loss makes more sense because it gives us the minimum distance between predicted and actual probabilities for different values of k.

Assignment 2. Linear regression and ridge regression

Setting up the data

We read the dataset parkinsons.csv into R

Scaling: We scale the data because we want to be able to easily compare different independent variables and their contribution to the model.

Split the data: training and test data is separated in a 60-40 split

Inspect the training data: We then inspect the data to understand any underlying correlations between independent variables. For the sake of keeping the report concise, these plots have been given in the appendix. But upon observation, there seems to be a case of high covariance among the Jitter and Shimmer type variables.

Linear Regression

Fit the model: We now fit the following model: motor UPDRS = 0 + Jitter... + Jitter.Abs. + Jitter.RAP + Jitter.PPQ5 + Jitter.DDP + Shimmer + Shimmer.dB. + Shimmer.APQ3 + Shimmer.APQ5 + Shimmer.APQ11 + Shimmer.DDA + NHR + HNR + RPDE + DFA + PPE

Note that we have set the intercept to 0 because we have scaled the data earlier.

```
##                       [,1]
## Jitter...       0.181064842
## Jitter.Abs.    -0.169829827
## Jitter.RAP     -5.098808872
## Jitter.PPQ5    -0.071777454
## Jitter.DDP      5.079056168
## Shimmer         0.590992074
## Shimmer.dB.    -0.172859577
## Shimmer.APQ3   32.213851526
## Shimmer.APQ5   -0.386846497
## Shimmer.APQ11   0.310255686
## Shimmer.DDA   -32.529914655
## NHR            -0.186754802
## HNR            -0.239777026
## RPDE            0.003958015
## DFA            -0.277038107
## PPE             0.229005610
```

Interpretation of results:

Looking at the magnitudes of the coefficients, we can see that the variables: Shimmer.APQ3 & Shimmer.DDA have the largest coefficients followed by Jitter.RAP & Jitter.DDP rest all the features have a relatively small magnitudes and thus are less significant, but to truly understand the significance of each coefficient, we need additional data. To avoid manually calculating the t-values, p-values and other statistics that can help us answer this question, we'll re-run the linear regression model using the lm() in order to verify our results.

```
##
## Call:
## lm(formula = fit1, data = train_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0119 -0.7270 -0.1018  0.7384  2.1959
##
## Coefficients:
```

```
##                  Estimate Std. Error t value Pr(>|t|)
## Jitter...         0.181065   0.144249   1.255 0.209481
## Jitter.Abs.      -0.169830   0.040851  -4.157 3.30e-05 ***
## Jitter.RAP       -5.098809  18.184783  -0.280 0.779196
## Jitter.PPQ5      -0.071777   0.084701  -0.847 0.396816
## Jitter.DDP        5.079056  18.188164   0.279 0.780069
## Shimmer           0.590992   0.205286   2.879 0.004015 **
## Shimmer.dB.      -0.172860   0.139380  -1.240 0.214983
## Shimmer.APQ3     32.213852  77.012847   0.418 0.675759
## Shimmer.APQ5     -0.386846   0.113713  -3.402 0.000677 ***
## Shimmer.APQ11     0.310256   0.062270   4.982 6.58e-07 ***
## Shimmer.DDA     -32.529915  77.012630  -0.422 0.672761
## NHR              -0.186755   0.045741  -4.083 4.55e-05 ***
## HNR              -0.239777   0.036565  -6.558 6.27e-11 ***
## RPDE              0.003958   0.022611   0.175 0.861052
## DFA              -0.277038   0.019888 -13.930  < 2e-16 ***
## PPE               0.229006   0.033264   6.885 6.84e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9366 on 3509 degrees of freedom
## Multiple R-squared:  0.1212, Adjusted R-squared:  0.1172
## F-statistic: 30.25 on 16 and 3509 DF,  p-value: < 2.2e-16
```

The first thing we check is that our manual calculation of the beta coefficients was correct. Upon further inspection we find that the resulting model has a low adjusted r^2 value which explains only ~12% of the variance of the predicted disease symptom score. The F-statistic (given that we have a moderately large number of observations, the resulting F-statistic » 1, provides sufficient cause to reject H_0) and low p-value for the model.

Shimmer.APQ3 & Shimmer.DDA, which we previously identified as significant due to their large magnitude, have very high standard errors along with low t-values, thus resulting in high chances of the null hypothesis H_0 (corresponding $\beta\ i\ = 0$) getting accepted. The corresponding higher p-values also indicate higher chances of H_0 getting accepted. The same inference can be made for Jitter.RAP and Jitter.DDP. This can also be easily verified if we check the correlation between these 2 pairs. They have a very high correlation thus resulting in the problem of multicollinearity in this fit. Note, we can't check for the multicollinearity problem directly using the vif() as the intercept is 0 in our model.

The other remaining variables have coefficients ~ 0 or p-values suggesting a high chance of H_0 getting accepted for their corresponding $\beta\ i$

Conclusion: We have a significant model but it is highly complex and over-fitted and we have encountered multicollinearity in fitting the data to the model. Thus the most significant features are as follows:

1. Jitter.Abs 2. Shimmer.APQ5 3. Shimmer.APQ11 4. NHR 5. HNR 6. DFA 7. PPE

Now that we have our model, we will use it to fit test data. We will also test the MSE to check the robustness of the model.

```
## [1] "\n Test MSE:  0.93"
```

```
## [1] "\n Train MSE:  0.87"
```

We cannot comment much on the overall MSE values other than that the MSE for the test set is not too different from the MSE for the training data.

Likelihood

Since it is given that the disease symptom score is normally distributed and a function of the voice parameters(θ), we'll be using the density function of the normal distribution to create the log-likelihood function for this data.

$$Y_i = \theta X_i + \epsilon_i$$
$$\epsilon_i = Y_i - \theta X_i$$
$$\epsilon_i \sim N(0, \sigma^2)$$
$$Y_i \sim N(\theta^T X_i, \sigma^2)$$

The density function will then become:

$$f(Y_i|\theta, \sigma^2) \sim \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(- \frac{(Y_i - \theta X_i)^2}{2\sigma^2}\Big)$$

In order to learn the parameters, we will be maximizing the likelihood in this exercise. For reducing analytical complexity and to avoid underflow due to floating point precision of scaled data we will be taking log of the likelihood.

$$\hat{\theta} = arg\ max_\theta\ p\Big(Y|X, \theta\Big)$$
$$L(\theta, \sigma^2) = \prod_{i=1}^{N} f(Y_i|\theta, \sigma^2) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(- \frac{(Y_i - \theta X_i)^2}{2\sigma^2}\Big)$$
$$\log L(\theta, \sigma^2) = \sum_{i=1}^{n} -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}(Y_i - \theta X_i)^2$$

We write the log-likehood() as follows in R:

```
log_likelihood <- function(theta, Y, X){
  X <- as.matrix(X)
  Y <- as.matrix(Y)
  n <- nrow(X)
  beta <- theta[1:ncol(X)]
  sigma2 <- theta[ncol(X)+1]^2
  err <- Y - X %*% beta
  loglik  <- -(n/2)*log(2*pi)-(n/2)*log(sigma2) - ((t(err) %*% err)/ (2*sigma2))
  return(-loglik)
}
```

We will later use the negative value of the function because optimization will be to minimize the negative of the log-likelihood. We will quickly run a test on the likelihood function to ensure the correctness of our approach

```
##          [,1]
## [1,] 236819.8
```

```
##                         OLS         MLE
## Jitter...       0.181064842  0.23613557
## Jitter.Abs.    -0.169829827  0.04460352
## Jitter.RAP     -5.098808872 -0.12265843
## Jitter.PPQ5    -0.071777454 -0.04721460
```

```
## Jitter.DDP      5.079056168 -0.07299552
## Shimmer         0.590992074  1.11533605
## Shimmer.dB.    -0.172859577 -0.31437808
## Shimmer.APQ3   32.213851526 -0.26201527
## Shimmer.APQ5   -0.386846497 -0.41379280
## Shimmer.APQ11   0.310255686  0.23441989
## Shimmer.DDA   -32.529914655 -0.27992390
## NHR            -0.186754802 -0.25238788
## HNR            -0.239777026 -0.17119833
## RPDE            0.003958015 -0.03447943
## DFA            -0.277038107 -0.26939918
## PPE             0.229005610  0.15280012
```

The coefficients estimated using the MLE method are compared to our previous model, huge differences lie in the largest coefficients from our OLS method i.e. Shimmer.APQ3 and Shimmer.DDA. We have already concluded earlier that due to high p-values, these coefficients were not significant. We can see that MLE estimates have reduced their overall magnitude significantly.

Ridge Regression

We already concluded earlier that we encountered multicollinearity in our fitted model. Hence we now perform Ridge regression by computing a L2 penalty and adding the same to the log-likelihood function

$$min_\theta = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \theta^T X_i\right)^2 + \lambda\sum_{j=1}^{p}\theta_j^2$$

So now we add another function for adding the penalty term to the negative loglikelihood

We also add the function to optimize the ridge() for optimal θ & σ values with user specified λ values

Next we specify the function to calculate the degrees of freedom for the Ridge regression. The degrees of freedom are calculated as the trace of the following matrix:

$$trace(X(X'X + \lambda I)^{-1}X')$$

The function for calculating the degrees of freedom is as follows:

```
DF <- function(X, lambda) {
  df = sum(diag(((X %*% solve((t(X) %*% X) + lambda*diag(ncol(X)))) %*% t(X))))
  return(df)
}
```

Results of Ridge Regression

We now test our ridge regression outputs for, $\lambda = 1, 100, 1000$

```
##                        OLS         MLE  rid_lambda1 rid_lamda100
## Jitter...       0.181064842  0.23613557  0.172942533   0.04704537
## Jitter.Abs.    -0.169829827  0.04460352 -0.168815148  -0.12992043
## Jitter.RAP     -5.098808872 -0.12265843 -0.012587719   0.01585414
## Jitter.PPQ5    -0.071777454 -0.04721460 -0.067754913  -0.02608535
## Jitter.DDP      5.079056168 -0.07299552 -0.003691456   0.01591744
## Shimmer         0.590992074  1.11533605  0.533380369   0.03747447
## Shimmer.dB.    -0.172859577 -0.31437808 -0.144892782   0.02142729
## Shimmer.APQ3   32.213851526 -0.26201527 -0.148788575  -0.07633105
## Shimmer.APQ5   -0.386846497 -0.41379280 -0.373721369  -0.10037407
## Shimmer.APQ11   0.310255686  0.23441989  0.311086024   0.22442929
```

```
## Shimmer.DDA   -32.529914655 -0.27992390 -0.151950352  -0.07634278
## NHR            -0.186754802 -0.25238788 -0.184541395  -0.14192851
## HNR            -0.239777026 -0.17119833 -0.238339550  -0.18287429
## RPDE            0.003958015 -0.03447943  0.004991525   0.02780841
## DFA            -0.277038107 -0.26939918 -0.276044422  -0.24228047
## PPE             0.229005610  0.15280012  0.228529945   0.21601871
##               rid_lambda1000 rid_lambda0
## Jitter...        0.005720011  0.23613557
## Jitter.Abs.     -0.040720790  0.04460352
## Jitter.RAP      -0.003442182 -0.12265843
## Jitter.PPQ5     -0.006666166 -0.04721460
## Jitter.DDP      -0.003434516 -0.07299552
## Shimmer          0.005692635  1.11533605
## Shimmer.dB.      0.012879479 -0.31437808
## Shimmer.APQ3    -0.017121068 -0.26201527
## Shimmer.APQ5    -0.009533944 -0.41379280
## Shimmer.APQ11    0.071690425  0.23441989
## Shimmer.DDA     -0.017126453 -0.27992390
## NHR             -0.036802641 -0.25238788
## HNR             -0.078573484 -0.17119833
## RPDE             0.045321388 -0.03447943
## DFA             -0.126036041 -0.26939918
## PPE              0.104454361  0.15280012
```

On comparing the coefficient values, we find that as the value of λ increases our coefficients for the model start approaching 0, but they never actually become = 0 as would be the case in LASSO regression. When we reach the case of $\lambda = 1000$, there are only 4 remaining coefficients with significant magnitude which is a significant improvement over our OLS and MLE solution.

We now check for improvement in the output of the model by calculating the predicted values for the training and test data and checking the following two parameters:

1. MSE
2. Degrees of Freedom

```
## [1] "Training OLS MSE: 0.873193058836616"
```

```
##                            [,1]
## MSE_lambda1_train    0.8732770
## MSE_lambda100_train  0.8790612
## MSE_lambda1000_train 0.9156267
```

```
## [1] "\n Test OLS MSE: 0.929491066167785"
```

```
##                           [,1]
## MSE_lambda1_test    0.9290354
## MSE_lambda100_test  0.9263128
## MSE_lambda1000_test 0.9479166
```

For the 3 different value of λ that we have chosen, we see that for $\lambda = 100$ the test MSE is the lowest of the 3 choices, hence that would be our choice.

Next we examine the calculated Degrees of Freedom after adding the Ridge penalty

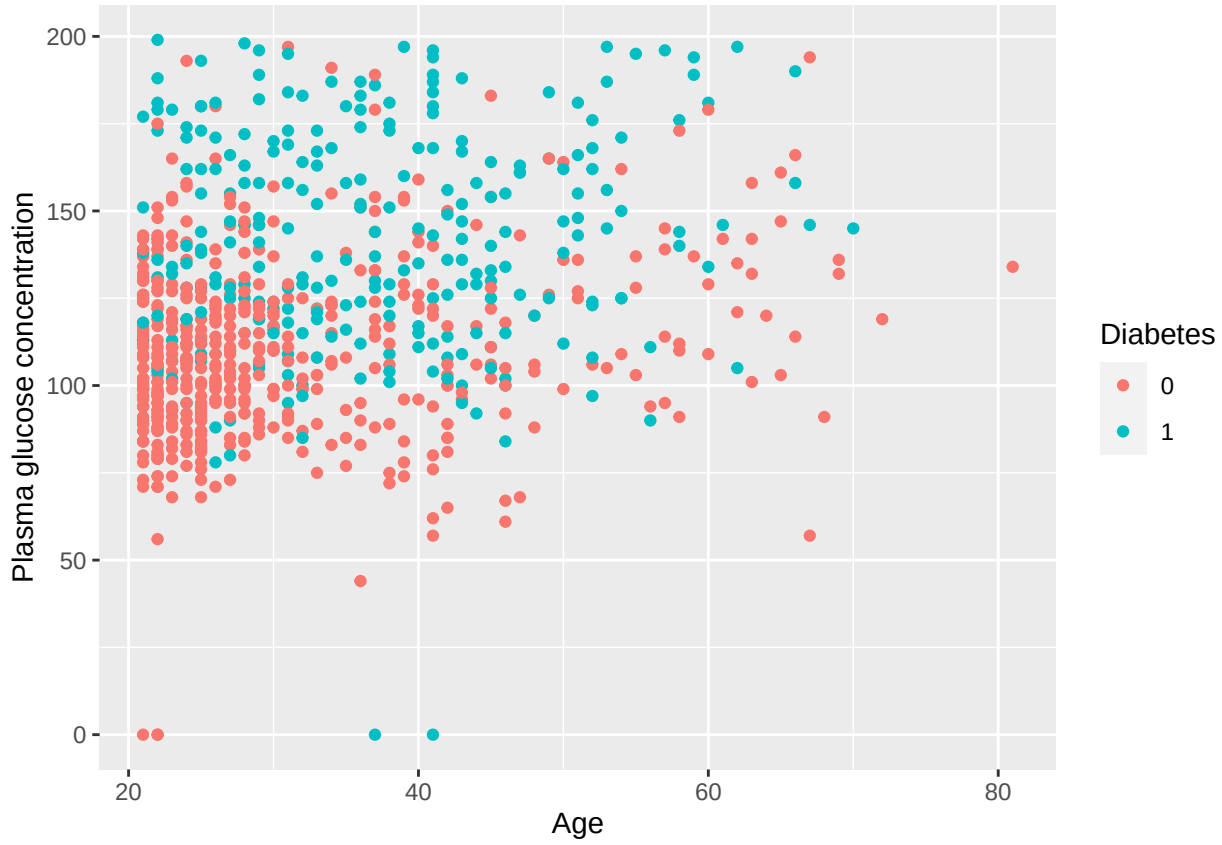
```
##                    [,1]
## DF_lambda1    13.862811
## DF_lambda100   9.939085
## DF_lambda1000  5.643351
```

```
##                       [,1]
## DF_lambda1     13.780433
## DF_lambda100    9.182898
## DF_lambda1000   4.821676
```

We immediately observe a huge loss in degrees of freedom for our data. This obviously flows from the 'shrinkage' in coefficients due to the applied Ridge penalty. Thus, our parameters are heavily constrained and hence the DFs is reflective of the remaining features after shrinkage.

Assignment 3. Logistic regression and basis function expansion

Initial Plot of Diabetes by Age vs Plasma Glucose:



No, it's not enough to classify by using standard logistic regression model that only uses age and plasma glucose concentration to predict whether a person has diabetes or not. Higher (>100) Plasma glucose concentration seems to suggest diabetes but its hard to predict if concentration is between 100 to 150. A candidate's Age seems to be a poor indicator of diabetes because even though at a younger age there a higher probability of not having diabetes, but in older age brackets, its seems harder to classify.

Model training:

```
##
## Call:
## glm(formula = Diabetes ~ Glu + Age, family = binomial(link = "logit"),
##     data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3303  -0.7775  -0.5095   0.8370   3.1617
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -5.897858   0.462450  -12.75  < 2e-16 ***
## Glu          0.035582   0.003288   10.82  < 2e-16 ***
## Age          0.024502   0.007379    3.32 0.000899 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 991.38  on 766  degrees of freedom
## Residual deviance: 796.49  on 764  degrees of freedom
## AIC: 802.49
##
## Number of Fisher Scoring iterations: 4
```

Prediction head():

```
##          1          2          3          4          5          6
## 0.10776306 0.80183292 0.09828009 0.44657875 0.26206931 0.07688932
```

Confusion Matrix:

```
##            Predictedvalue
## Actualvalue FALSE TRUE
##           0   436   64
##           1   140  127
```

Threshold [r=0.5] head():

```
## 1 2 3 4 5 6
## 0 1 0 0 0 0
```

Probabilistic equation of the estimated model:

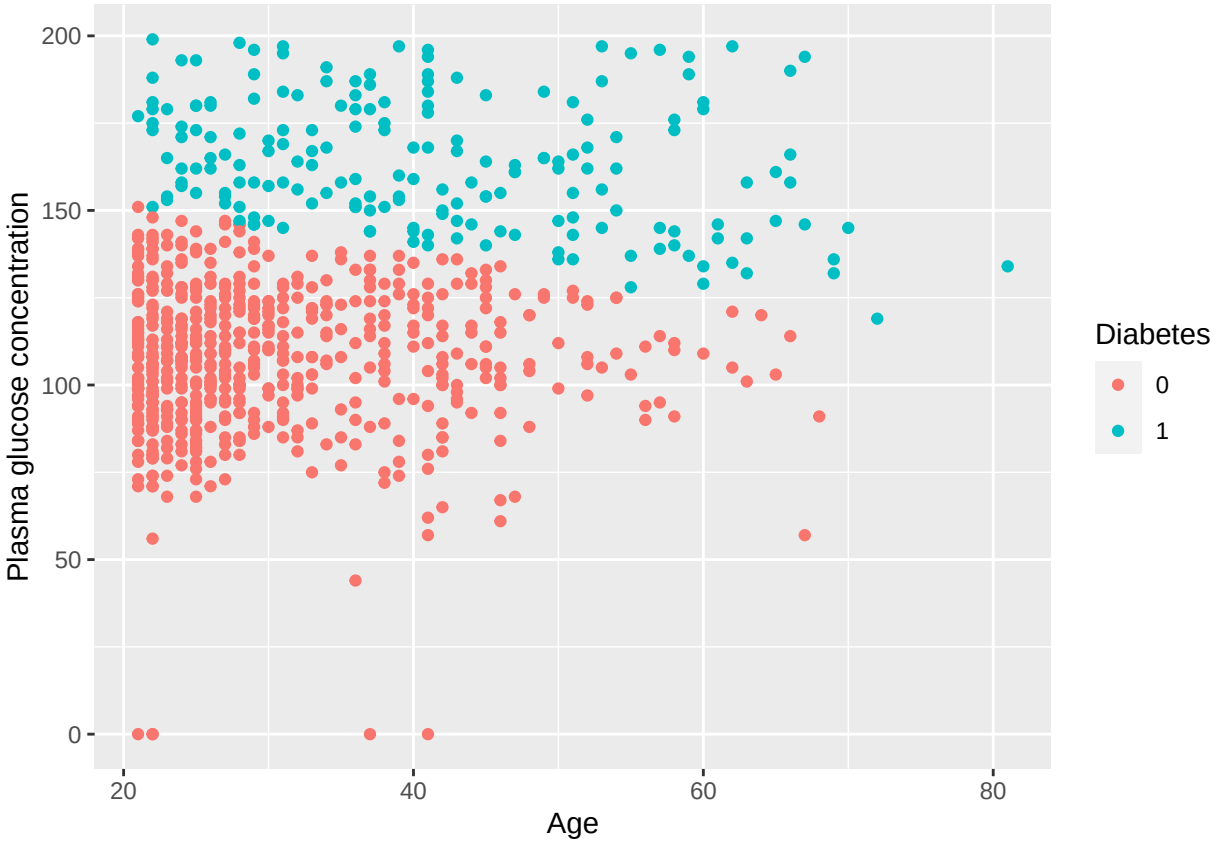
$$P(Diabetes|Glu, Age) = \frac{exp(-5.9 + 0.035 * Glu + 0.025 * Age)}{1 + exp(-5.9 + 0.035 * Glu + 0.025 * Age)}$$

$$; where\ logit\ is\ log\Big(\frac{P(Diabetes = 1|Glu, Age}{1 - P(Diabetes = 1|Glu, Age))}\Big) = -5.9 + 0.035 * Glu + 0.025 * Age$$

Mean misclassification error:

```
## [1] 0.266
```

Plot:



Quality of classification is just satisfactory as model accuracy is about 73.4%, which we think that practically is not that good in this type of problem because we are trying to classify a health related condition. A misclassification can lead to complications with both kind of patients who have diabetes and those who don't. Hence, we should be looking for a model with higher accuracy in prediction.

Equation of decision boundary:

The decision boundary equation can be determined by solving the equation which gives us all the points where the probability for both classes is the same

$$P(X) = 1 - P(X)$$

Therefore, the slope and intercept of the decision boundary can be calculated as follows:

```
slope <- -model_1$coefficients[1]/model_1$coefficients[2]
round(slope,2)
```

```
## (Intercept)
##      165.75
```

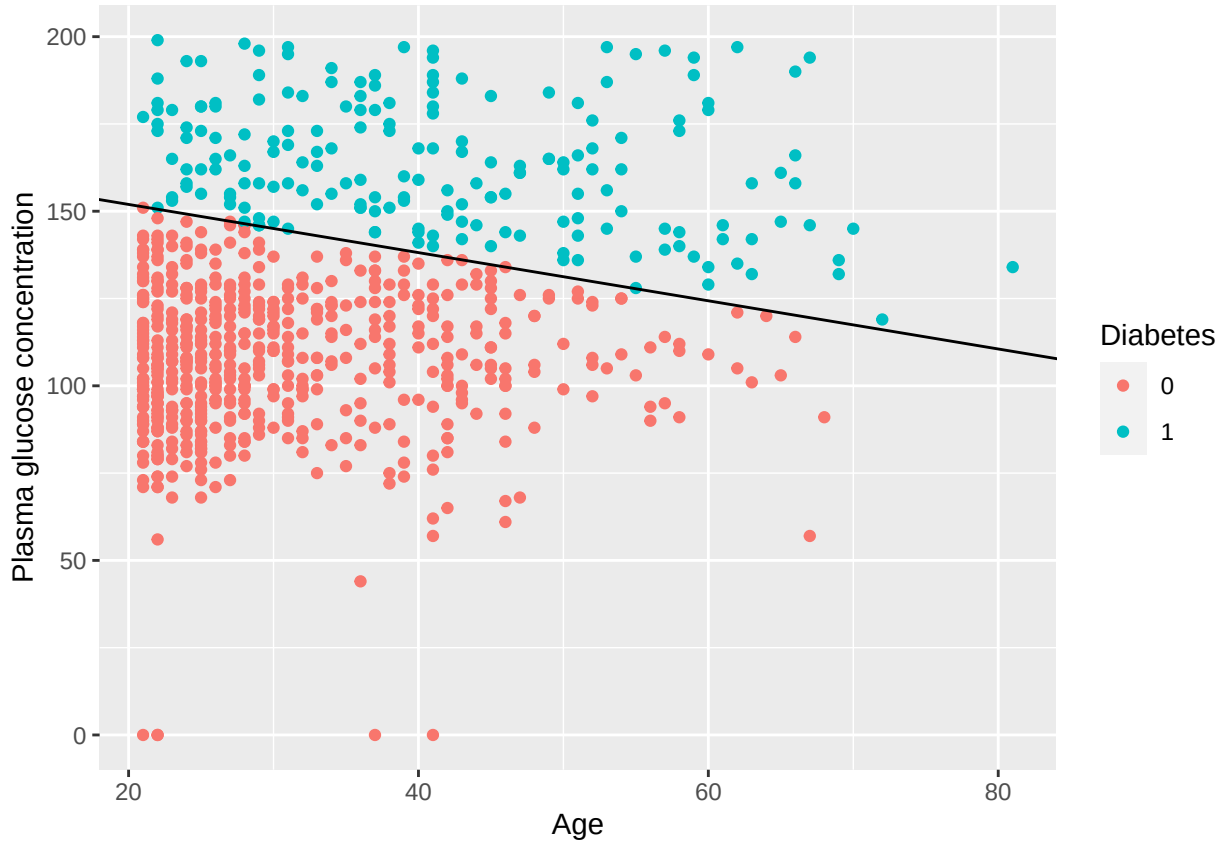
```
intercept <- -model_1$coefficients[3]/model_1$coefficients[2]
round(intercept,2)
```

```
##   Age
## -0.69
```

So, finally the reported decision boundary is:

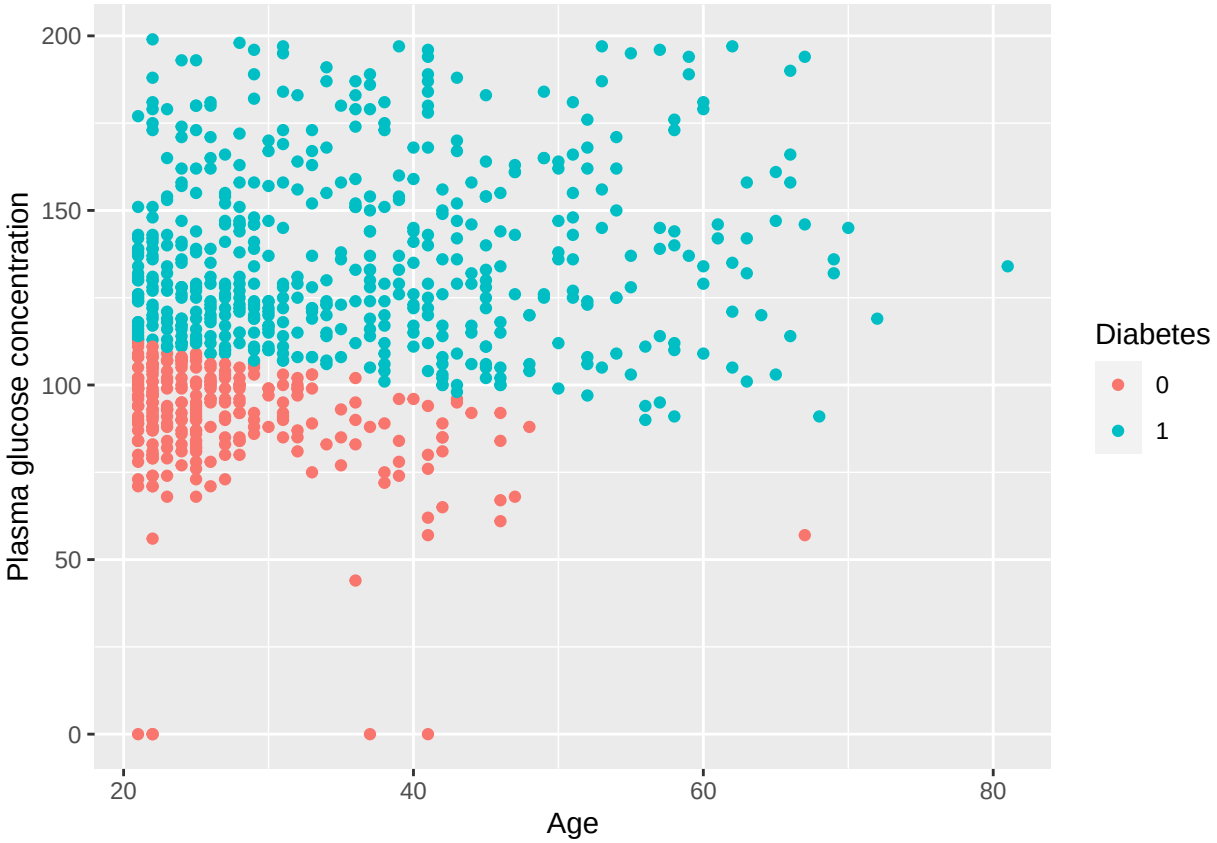
$$y = -0.69x + 165.75$$

Plot:



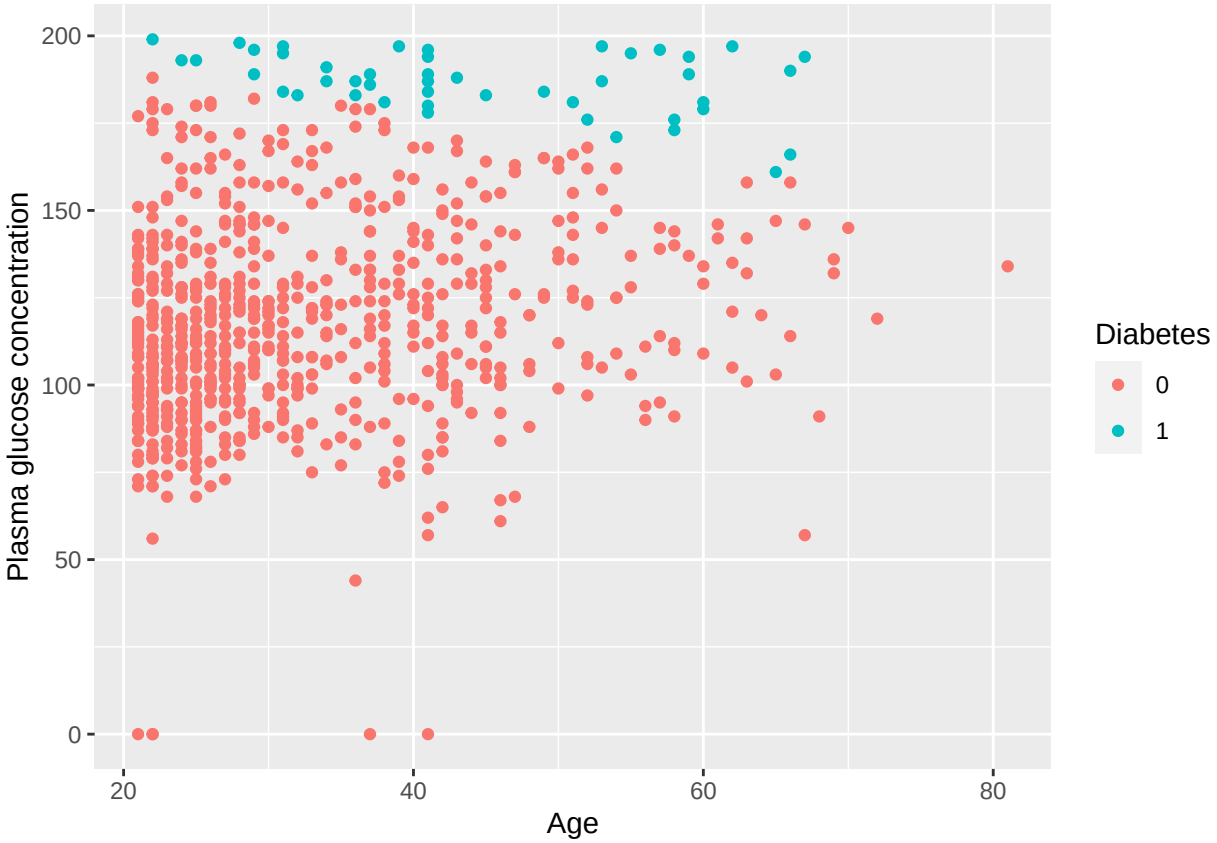
The decision boundary seems to catch the data distribution perfectly using slope and intercept from the equation of decision boundary.

For threshold $r = 0.2$:




```
##            Predictedvalue
## Actualvalue FALSE TRUE
##           0   238  262
##           1    25  242
```

For threshold $r = 0.8$:




```
##            Predictedvalue
## Actualvalue FALSE TRUE
##           0   490   10
##           1   231   36
```

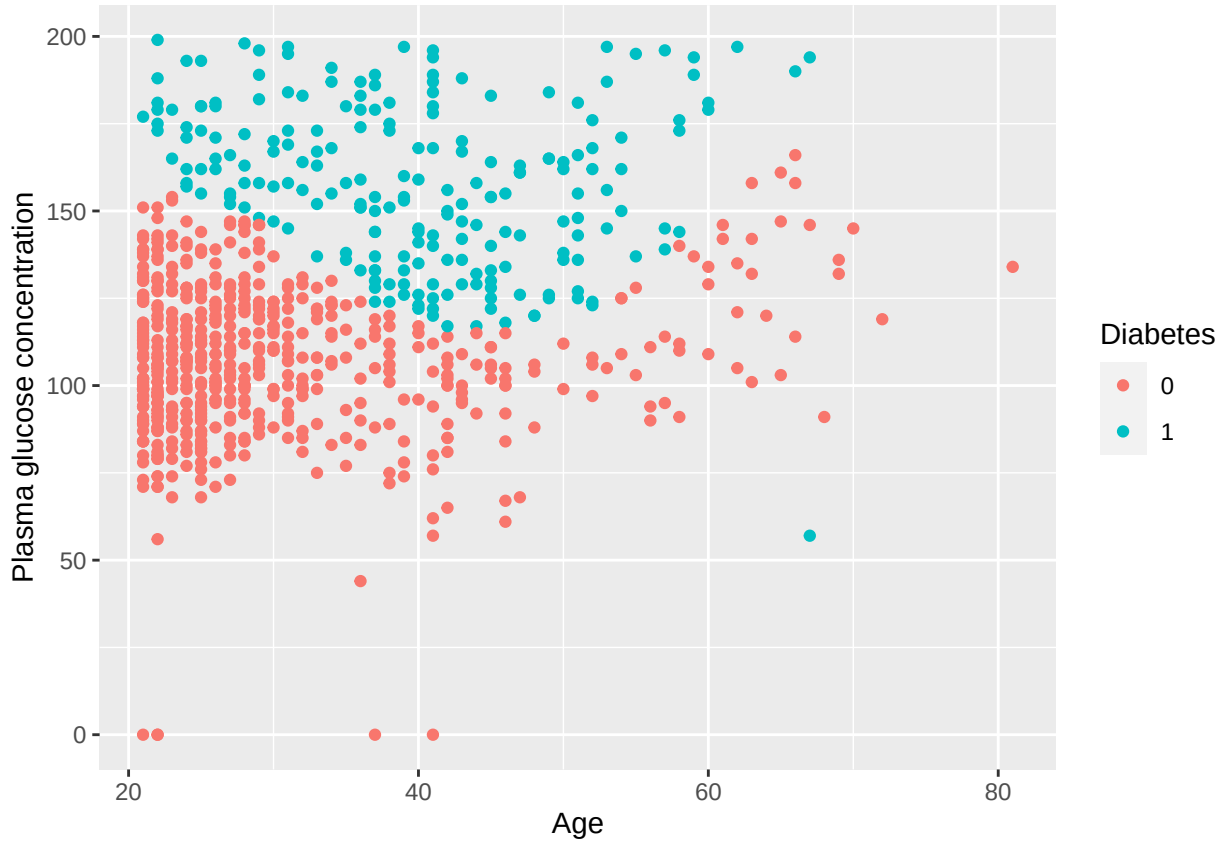
By changing the threshold we can clearly see that the classification results are not desirable. for r=0.2, we are able to classify positive diabetes much better, but we are also getting a lot of false positives. On the other hand, with r = 0.8, our model classifies true negatives well, but misclassification for the positive diabetes cases is very high.

Model training with new features:

```
##
## Call:
## glm(formula = Diabetes ~ Glu + Age + z1 + z2 + z3 + z4 + z5,
##     family = binomial(link = "logit"), data = df2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2874  -0.7267  -0.4273   0.7489   2.5260
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.279e+00  1.129e+00  -8.217  < 2e-16 ***
## Glu          3.772e-02  9.473e-03   3.981 6.85e-05 ***
## Age          1.453e-01  2.072e-02   7.014 2.32e-12 ***
## z1           1.266e-08  5.610e-09   2.257  0.02402 *
## z2          -1.760e-07  7.638e-08  -2.304  0.02122 *
```

```
## z3           8.424e-07  3.439e-07   2.450  0.01430 *
## z4          -1.682e-06  6.317e-07  -2.662  0.00776 **
## z5           8.045e-07  4.056e-07   1.983  0.04732 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 991.38  on 766  degrees of freedom
## Residual deviance: 741.07  on 759  degrees of freedom
## AIC: 757.07
##
## Number of Fisher Scoring iterations: 5
```

Plot:



Misclassification:

```
## [1] 0.246
```

The misclassification error of the model hasn't improved that drastically compared to previous model (~2% improvement). This new model is slightly more accurate due because we used the basis expansion trick. Basic expansion trick changed the shape of decision boundary to a bell shape because now the model is more flexible but the accuracy didn't improve too much because not all learned features are significant.

Appendix I: CODE

```
library(kknn)
library(ggplot2)
library(dplyr)
library(stats)
library(plotly)
library(car)
knitr::opts_chunk$set(echo = TRUE)
#Assignment 1


# read csv file from working directory
data <- read.csv("data/optdigits.csv")

# give data set ID column
data[,66] <- seq_len(nrow(data))

# give names 1:64 + Truth + ID
names <- c(as.character(c(1:64)), "Truth", "ID")
colnames(data) <- names


# split the data into train (50%) validation (25%) and test (25%)
n <- dim(data)[1] # equals number of rows

## train
set.seed(12345) # according to lecture 1 a
id <- sample(1:n, floor(n*0.5))
train <- data[id,]

## validation
id1 <- setdiff(1:n, id)
set.seed(12345)
id2 <- sample(id1 ,floor(n*0.25))
valid <- data[id2,]

## test
id3 <- setdiff(id1, id2)
test <- data[id3,]


# calculate the 30 nearest neighbor classifier
fit30_train <- kknn(as.factor(train$Truth)~ ., train, train, k = 30, kernel="rectangular")
fit30_test <- kknn(as.factor(train$Truth)~ ., train, test, k = 30, kernel="rectangular")

# confusion table for train data
confusion_train <- table(true=train[,65], prediction=fit30_train$fitted.values)
print(confusion_train)
misclassifications_train <- rowSums(confusion_train)-diag(confusion_train)

# confusion table for test data
confusion_test <- table(true=test[,65], prediction=fit30_test$fitted.values)
```

```
print(confusion_test)
misclassifications_test <- rowSums(confusion_test)-diag(confusion_test)

# misclassification error train and test respectively
error_train <- sum(confusion_train)-sum(diag(confusion_train))
error_test <- sum(confusion_test)-sum(diag(confusion_test))

error_train
error_test
# misclassification rate train and test respectively
accuracy_train <- 1/sum(confusion_train)*(sum(diag(confusion_train)))
misclassification_error_train <- 1-accuracy_train

accuracy_test <- 1/sum(confusion_test)*(sum(diag(confusion_test)))
misclassification_error_test <- 1-accuracy_test

# misclassification rate for test data
print(accuracy_train)
print(accuracy_test)
# take out index with 8's
is_eight <- which(train[,65] == 8)

# add an index column
data_new <- seq_len(nrow(fit30_train$prob))
prob_eight_i <- data.frame(cbind(fit30_train$prob, data_new))
colnames(prob_eight_i) <- 0:10 # 11 is index

# sort out any high and low probability
prob_eight <- as.data.frame(prob_eight_i[is_eight,])
prob_eight_low <- prob_eight %>% arrange(prob_eight[,9])
prob_eight_high <- prob_eight %>% arrange(desc(prob_eight[,9]))

# rearrange columns to 8x8
## for high probability eights
eight1 <- matrix(unlist(train[(prob_eight_high[1, 11]), 1:64]), nrow = 8, byrow = TRUE)
eight2 <- matrix(unlist(train[(prob_eight_high[2, 11]), 1:64]), nrow = 8, byrow = TRUE)
## for low probability eights
eight3 <- matrix(unlist(train[(prob_eight_low[1, 11]), 1:64]), nrow = 8, byrow = TRUE)
eight4 <- matrix(unlist(train[(prob_eight_low[2, 11]), 1:64]), nrow = 8, byrow = TRUE)
eight5 <- matrix(unlist(train[(prob_eight_low[3, 11]), 1:64]), nrow = 8, byrow = TRUE)

heatmap1 <- heatmap((eight1), Colv=NA, Rowv=NA, xlab = "Columns", ylab = "Rows")
heatmap2 <- heatmap(eight2, Colv=NA, Rowv=NA)
heatmap3 <- heatmap(eight3, Colv=NA, Rowv=NA)
heatmap4 <- heatmap(eight4, Colv=NA, Rowv=NA)
heatmap5 <- heatmap(eight5, Colv=NA, Rowv=NA)




fit_t <- vector(mode = "list", length = 30)
fit_v <- vector(mode = "list", length = 30)
```

```r
confusion_train <- vector(mode = "list", length = 30)
confusion_valid <- vector(mode = "list", length = 30)
misclassification_error_train <- matrix(NA, nrow = 1, ncol = 30)
misclassification_error_valid <- matrix(NA, nrow = 1, ncol = 30)

for (i in 1:30) {
  fit_t[[i]] <- kknn(as.factor(train$Truth)~ ., train[,], train[,], k = i, kernel="rectangular")
  fit_v[[i]] <- kknn(as.factor(train$Truth)~ ., train[,], valid[,], k = i, kernel="rectangular")
  confusion_train[[i]] <- table(true=train[,65], prediction=fit_t[[i]]$fitted.values)
  confusion_valid[[i]] <- table(true=valid[,65], prediction=fit_v[[i]]$fitted.values)
  misclassification_error_train[i] <- 1-1/sum(confusion_train[[i]])*(sum(diag(confusion_train[[i]])))
  misclassification_error_valid[i] <- 1-1/sum(confusion_valid[[i]])*(sum(diag(confusion_valid[[i]])))
}

plot(x = 1:30, y = misclassification_error_train*100, ylim = c(0,5),
     xlab = "K", ylab = "Misclassification error (%)", "o", col = "blue")
points(x = 1:30, y = misclassification_error_valid*100, "o",col = "red")
legend(x = "topleft", legend=c("Training data", "Validation data"), lty = c(1, 1), col=c("blue", "red"))


fit_best <- kknn(as.factor(train$Truth)~ ., train, test, k = 5, kernel="rectangular")
confusion_test <- table(true=test[,65], prediction=fit_best$fitted.values)
misclassification_error_test8 <- 1-1/sum(confusion_test)*(sum(diag(confusion_test)))
misclassification_error_test8
# calculate the cross entropy loss
cross_entropy_v <- matrix(NA, nrow = 1, ncol = 30)
for (i in 1:30) {
  cross_entropy_v[i] <- -sum(log(fit_v[[i]]$prob[cbind(seq_len(nrow(valid)),(valid[,65]+1))] + 10^(-15))
}

plot(x = 1:30, cross_entropy_v, "o", pch = 1, xlab = "K", ylab = "Cross entropy")




#Assignment 2

parkinson_data <- read.csv("data/parkinsons.csv")
parkinson_scaled <- scale(parkinson_data)
n=dim(parkinson_scaled)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.6))
train=parkinson_scaled[id,]
test=parkinson_scaled[-id,]
fit1 <- motor_UPDRS ~ 0 + Jitter... + Jitter.Abs. + Jitter.RAP + Jitter.PPQ5 + Jitter.DDP + Shimmer + Sh
X_train <- model.matrix(fit1, as.data.frame(train))
Y_train <- train[,5]

#OLS using QR decomp
Q <- qr.Q(qr(X_train))
R <- qr.R(qr(X_train))

betas_qr <- solve(R) %*% t(Q) %*% Y_train
```

```
rm(Q,R)

print(betas_qr)
train_df <- as.data.frame(train)

lm_train1 <- lm(fit1, train_df)
summary(lm_train1)
rm(train_df)
X_test <- model.matrix(fit1, as.data.frame(test))

predicted_test <- X_test %*% betas_qr
predicted_train <- X_train %*% betas_qr

Y_test <- test[,5]
Y_train <- train[,5]

MSE_train_OLS <- mean((Y_train - predicted_train)^2)
MSE_test_OLS <- mean((Y_test - predicted_test)^2)

#summary(Y_test)
print(paste("\n Test MSE: ", round(MSE_test_OLS,2)))

#summary(Y_train)
print(paste("\n Train MSE: ", round(MSE_train_OLS,2)))

rm(X_test, X_train, Y_test, Y_train)
log_likelihood <- function(theta, Y, X){
  X <- as.matrix(X)
  Y <- as.matrix(Y)
  n <- nrow(X)
  beta <- theta[1:ncol(X)]
  sigma2 <- theta[ncol(X)+1]^2
  err <- Y - X %*% beta
  loglik  <- -(n/2)*log(2*pi)-(n/2)*log(sigma2) - ((t(err) %*% err)/ (2*sigma2))
  return(-loglik)
}
theta = rep(1, 17)
theta = as.matrix(theta)
Y <- train[,5]
X <- train[,7:22]
#dispersion = 1

log_likelihood(theta, Y, X)
theta_0 <- rep(1, 17)

llcheck <- optim(par = theta_0, fn = log_likelihood, method = "BFGS", X = X, Y = Y, hessian = TRUE)
llbetas <- as.matrix(llcheck$par)

beta_compare <- cbind(betas_qr, llbetas[1:16,])
colnames(beta_compare) <- c("OLS", "MLE")
rm(theta, theta_0, X, Y, llcheck, llbetas)

print(beta_compare)
```

```r
#adding penalty to loglikelihood
ridge <- function(theta, lambda, Y, X) {
  lgllh <- log_likelihood(theta, Y, X)
  X <- as.matrix(X)
  beta <- theta[1:ncol(X)]
  penalty <- lambda * (t(beta) %*% beta)
  ridge <- lgllh + penalty
  return(ridge)
}
#optimization function for ridge
ridgeopt <- function(ridge_input, theta) {
  rid_op <- optim(par = theta, fn = ridge, method = "BFGS", X=X, Y=Y, lambda = lambda)
  return(rid_op$par)
}
DF <- function(X, lambda) {
  df = sum(diag(((X %*% solve((t(X) %*% X) + lambda*diag(ncol(X)))) %*% t(X))))
  return(df)
}
theta = rep(1, 17)
Y <- train[,5]
X <- train[,7:22]
lambda = 1

ridge_test <- ridge(theta, lambda, Y, X)

lambda = 1
ridopt_test <- ridgeopt(ridge_test, theta)
ridopt_test <- as.matrix(ridopt_test[1:16])
beta_compare <- cbind(beta_compare, ridopt_test)

lambda = 100
ridopt_test <- ridgeopt(ridge_test, theta)
ridopt_test <- as.matrix(ridopt_test[1:16])
beta_compare <- cbind(beta_compare, ridopt_test)

lambda = 1000
ridopt_test <- ridgeopt(ridge_test, theta)
ridopt_test <- as.matrix(ridopt_test[1:16])
beta_compare <- cbind(beta_compare, ridopt_test)

lambda = 0
ridopt_test <- ridgeopt(ridge_test, theta)
ridopt_test <- as.matrix(ridopt_test[1:16])
beta_compare <- cbind(beta_compare, ridopt_test)

colnames(beta_compare) <- c("OLS", "MLE", "rid_lambda1", "rid_lamda100", "rid_lambda1000", "rid_lambda0

rm(ridge_test, theta, Y, X, lambda)
print(beta_compare)
#predicted values for ridgeopt() using lambda = 1,100,1000

X_train <- as.matrix(train[,7:22])
X_test <- as.matrix(test[,7:22])
```

```r
beta_compare <- as.matrix(beta_compare)

#train

predict_lambda1 <- X_train %*% beta_compare[,3]
predict_lambda100 <- X_train %*% beta_compare[,4]
predict_lambda1000 <- X_train %*% beta_compare[,5]

predict_ridge_train <- as.data.frame(cbind(predict_lambda1,predict_lambda100,predict_lambda1000))
names(predict_ridge_train) <- c("lambda1", "lambda100", "lambda1000")

rm(predict_lambda1, predict_lambda100, predict_lambda1000)

#test

predict_lambda1 <- X_test %*% beta_compare[,3]
predict_lambda100 <- X_test %*% beta_compare[,4]
predict_lambda1000 <- X_test %*% beta_compare[,5]

predict_ridge_test <- as.data.frame(cbind(predict_lambda1,predict_lambda100,predict_lambda1000))
names(predict_ridge_test) <- c("lambda1", "lambda100", "lambda1000")

rm(predict_lambda1, predict_lambda100, predict_lambda1000)

#MSE train

Y_train <- train[,5]
Y_test <- test[,5]

MSE_lambda1_train <- mean((Y_train - predict_ridge_train$lambda1)^2)
MSE_lambda100_train <- mean((Y_train - predict_ridge_train$lambda100)^2)
MSE_lambda1000_train <- mean((Y_train - predict_ridge_train$lambda1000)^2)

MSE_ridge_train <- rbind(MSE_lambda1_train,MSE_lambda100_train,MSE_lambda1000_train)
names(predict_ridge_test) <- c("lambda1", "lambda100", "lambda1000")

rm(MSE_lambda1_train,MSE_lambda100_train,MSE_lambda1000_train)

#MSE test

MSE_lambda1_test <- mean((Y_test - predict_ridge_test$lambda1)^2)
MSE_lambda100_test <- mean((Y_test - predict_ridge_test$lambda100)^2)
MSE_lambda1000_test <- mean((Y_test - predict_ridge_test$lambda1000)^2)

MSE_ridge_test <- rbind(MSE_lambda1_test,MSE_lambda100_test,MSE_lambda1000_test)
names(predict_ridge_test) <- c("lambda1", "lambda100", "lambda1000")

rm(MSE_lambda1_test,MSE_lambda100_test,MSE_lambda1000_test)

#DF train

DF_lambda1 <- DF(X_train, 1)
DF_lambda100 <- DF(X_train, 100)
```

```
DF_lambda1000 <- DF(X_train, 1000)

DF_lambda_train <- rbind(DF_lambda1, DF_lambda100, DF_lambda1000)

rm(DF_lambda1, DF_lambda100, DF_lambda1000)

#DF test

DF_lambda1 <- DF(X_test, 1)
DF_lambda100 <- DF(X_test, 100)
DF_lambda1000 <- DF(X_test, 1000)

DF_lambda_test <- rbind(DF_lambda1, DF_lambda100, DF_lambda1000)

rm(DF_lambda1, DF_lambda100, DF_lambda1000)

print(paste("Training OLS MSE:", MSE_train_OLS))
print(MSE_ridge_train)

print(paste("\n Test OLS MSE:", MSE_test_OLS))
print(MSE_ridge_test)
print(DF_lambda_train)
print(DF_lambda_test)
rm(list = ls())



# Assignment 3


data <- read.csv("data/pima-indians-diabetes.csv")
df <- as.data.frame(data)
colnames(df)<- c("Npreg", "Glu", "Bp", "Skin", "Serum",
                 "Bmi", "Ped", "Age", "Diabetes")
ggplot(df, aes(y = Glu, x = Age )) +
  geom_point(aes(color = factor(Diabetes))) +
  labs(
    x = "Age",
    y = "Plasma glucose concentration",
    color = "Diabetes"
    )

model_1 <- glm(Diabetes ~ Glu + Age,
               family = binomial(link = "logit"),
               data = df)
summary(model_1)
pred <- predict(model_1, df,  type = "response")
head(pred)
cm <- table(Actualvalue = df$Diabetes, Predictedvalue = pred > 0.5)
cm
pred_thresh <- ifelse(pred > 0.5, 1, 0)
head(pred_thresh)
mmce <- 1 - (sum(diag(cm))/sum(cm))
```

```
round(mmce,3)
  ggplot(df, aes(y = Glu, x = Age )) +
    geom_point(aes(color = factor(pred_thresh))) +
    labs(
      x = "Age",
      y = "Plasma glucose concentration",
      color = "Diabetes"
    )

slope <- -model_1$coefficients[1]/model_1$coefficients[2]
round(slope,2)

intercept <- -model_1$coefficients[3]/model_1$coefficients[2]
round(intercept,2)
  ggplot(df, aes(y = Glu, x = Age )) +
    geom_point(aes(color = factor(pred_thresh))) +
    geom_abline(slope = -0.69 , intercept =165.75 ) +
    labs(
      x = "Age",
      y = "Plasma glucose concentration",
      color = "Diabetes"
    )

pred_thresh_2 <- ifelse(pred > 0.2, 1, 0)
plot_thresh_2 <- ggplot(df, aes(y = Glu, x = Age )) +
    geom_point(aes(color = factor(pred_thresh_2))) +
    labs(
      x = "Age",
      y = "Plasma glucose concentration",
      color = "Diabetes"
    )
plot_thresh_2

table(Actualvalue = df$Diabetes, Predictedvalue = pred > 0.2)

pred_thresh_3 <- ifelse(pred > 0.8, 1, 0)
plot_thresh_3 <- ggplot(df, aes(y = Glu, x = Age )) +
    geom_point(aes(color = factor(pred_thresh_3))) +
    labs(
      x = "Age",
      y = "Plasma glucose concentration",
      color = "Diabetes"
    )

plot_thresh_3
table(Actualvalue = df$Diabetes, Predictedvalue = pred > 0.8)
# New features:
df2 <- df
df2$z1 <- df$Glu^4
df2$z2 <- df$Glu^3 * df$Age
df2$z3 <- df$Glu^2 * df$Age^2
df2$z4 <- df$Glu^1 * df$Age^3
df2$z5 <- df$Age^4
```

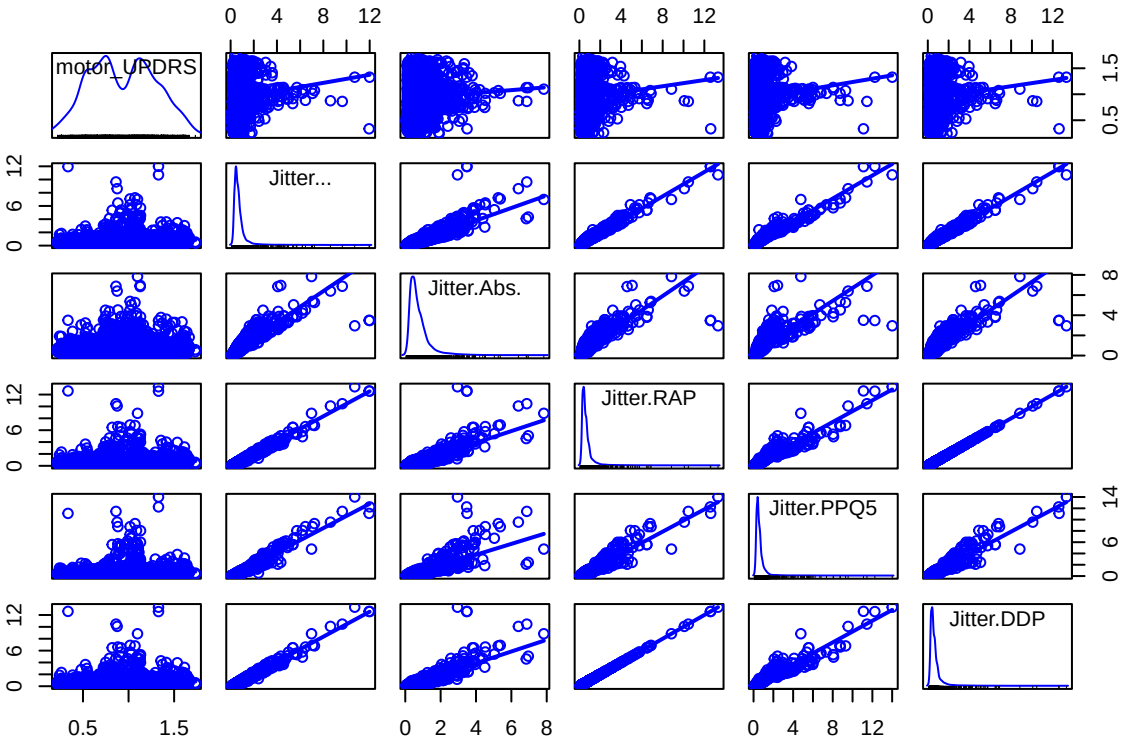
```r
# New model
model_2 <- glm(Diabetes ~ Glu + Age + z1 + z2 + z3 + z4 + z5,
               family = binomial(link = "logit"),
               data = df2)
summary(model_2)
# Prediction:
pred_4 <- predict(model_2, df2,  type = "response")
pred_thresh_4 <- ifelse(pred_4 > 0.5, 1, 0)
  ggplot(df2, aes(y = Glu, x = Age )) +
    geom_point(aes(color = factor(pred_thresh_4))) +
    labs(
      x = "Age",
      y = "Plasma glucose concentration",
      color = "Diabetes"
    )

# Confusion Matrix:
cm_4 <- table(Actualvalue = df2$Diabetes, Prictedvalue = pred_4 > 0.5)
mmce_4 <- 1 - (sum(diag(cm_4))/sum(cm_4))
round(mmce_4,3)

parkinson_data <- read.csv("data/parkinsons.csv")

parkinson_scaled <- scale(parkinson_data, center = FALSE)

#Partition into train & test (60/40)
n=dim(parkinson_scaled)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.6))
train=parkinson_scaled[id,]
test=parkinson_scaled[-id,]
##

rm(id, n)
train_df <- as.data.frame(train)

# check for dependencies with Jitter
train_check <- train_df[,5:22]
train_check <- train_check[,-2]
scatterplotMatrix(train_check[,1:6], diagonal = "boxplot", smooth = FALSE)

# check for dependencies with Shimmer
train_check1 <- train_check[,-2:-6]
scatterplotMatrix(train_check1[,1:7], diagonal = "boxplot", smooth = FALSE)

## check for dependencies with other variables
train_check2 <- train_check1[,-2:-7]
scatterplotMatrix(train_check2[,1:6], diagonal = "boxplot", smooth = FALSE)
```

Appendix II: Assignment 2 - Data inspection

Initial Data Observation for collinearity

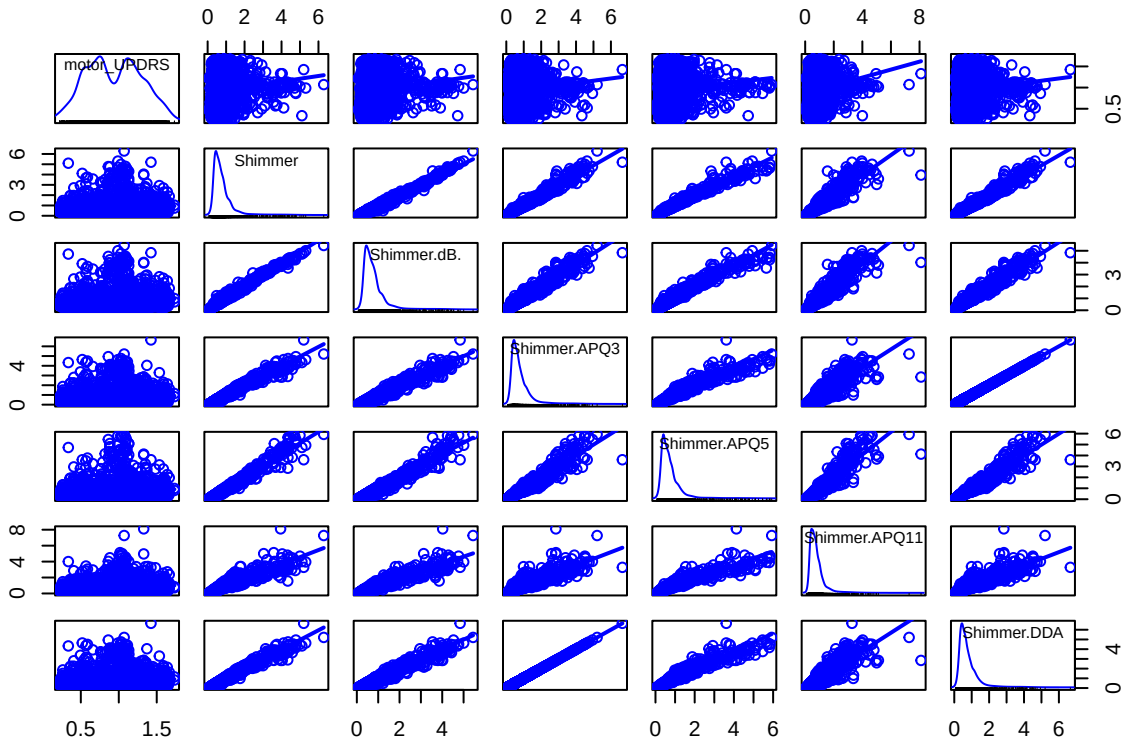
```
parkinson_data <- read.csv("data/parkinsons.csv")

parkinson_scaled <- scale(parkinson_data, center = FALSE)

#Partition into train & test (60/40)
n=dim(parkinson_scaled)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.6))
train=parkinson_scaled[id,]
test=parkinson_scaled[-id,]
##

rm(id, n)
train_df <- as.data.frame(train)

# check for dependencies with Jitter
train_check <- train_df[,5:22]
train_check <- train_check[,-2]
scatterplotMatrix(train_check[,1:6], diagonal = "boxplot", smooth = FALSE)
```

```
## Warning in applyDefaults(diagonal, defaults = list(method =
## "adaptiveDensity"), : unnamed diag arguments, will be ignored
```



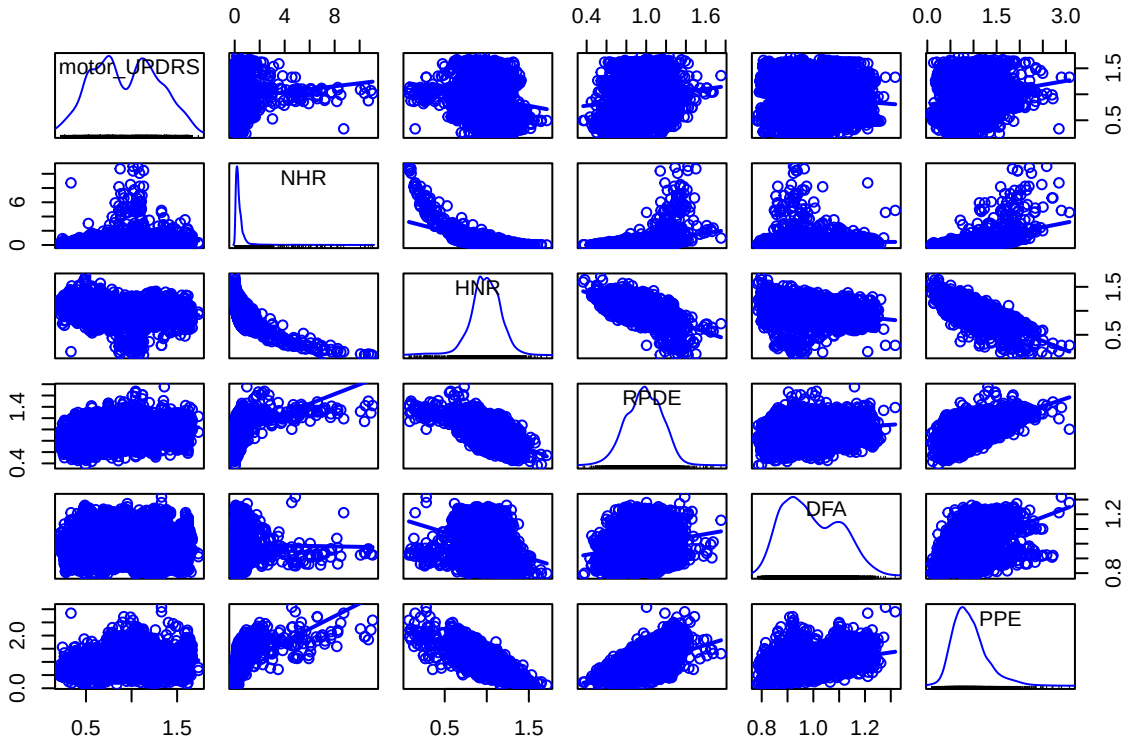

```
# check for dependencies with Shimmer
train_check1 <- train_check[,-2:-6]
scatterplotMatrix(train_check1[,1:7], diagonal = "boxplot", smooth = FALSE)
```

```
## Warning in applyDefaults(diagonal, defaults = list(method =
## "adaptiveDensity"), : unnamed diag arguments, will be ignored
```




```
## check for dependencies with other variables
train_check2 <- train_check1[,-2:-7]
scatterplotMatrix(train_check2[,1:6], diagonal = "boxplot", smooth = FALSE)
```

```
## Warning in applyDefaults(diagonal, defaults = list(method =
## "adaptiveDensity"), : unnamed diag arguments, will be ignored
```


motor_UPDRS
NHR
HNR
RPDE
DFA
PPE