Fake Bank Note Detection using Logistic Regression

Joshua Jose

2023-03-3

Aim

To identify whether a bank note is fake or not, based on features that were extracted from the images using wavelet transforms.

Data Description

Data were extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have 400x 400 pixels. Due to the object lens and distance to the investigated object gray-scale pictures with a resolution of about 660 dpi were gained. Wavelet Transform tool were used to extract features from images. The data set contains the variables(columns from left to right):

- Variance of wavelet Transformed image (continuous)
- Skewness of wavelet Transformed image (continuous)
- Kurtosis of wavelet Transformed image (continuous)
- Entropy of image (continuous)
- Class (integer) - (Fake bank note=1, Real bank note=0)

Theory

Since the dependent variable here is binary, we are planning on using Logistic Regression to identify the whether the bak note is fake or not. Logistic regression is a special type of the Generalized Linear Models.Here, the bivariate outcome Y has a Bernoulli distribution with parameter π (success probability $\pi \in (0, 1)$). Recall that $E[Y] = \pi$. The logit link function is

$$logit(E[Y]) = log\frac{E[Y]}{1 - E[Y]} = log\frac{\pi}{1 - \pi} = log\frac{P(Y = 1)}{P(Y = 0)}$$

Then, the logit of the mean (aka log odds) is modeled as a linear combination of the covariates (regressors) X, i.e., we have a linear predictor

$$logit(E[\mathbf{Y}]) = \mathbf{X}\beta$$

where β is a vector of unknown parameters. The maximum likelihood based approach is used for the parameter estimation.In order to find this estimator we'll have to use Fisher's Scoring method which can be seen in http://hua-zhou.github.io/teaching/biostatm280-2017spring/slides/18-newton/newton.html. Predicted probability can be obtained and it is

$$\hat{P}(Y_i = 1) = \frac{1}{1 + exp(-X_i^T\beta)}$$

Exploratory Data Analysis

Loading the required libraries

```
library(twinning)
library(ggplot2)
library(reshape2)
library(ROCR)
```

Loading the data

```
#The data does not have headers,so we have to specify that header=FALSE
data = read.csv("data_banknote_authentication.txt", header = FALSE)
colnames(data) = c("var_wave","skew_wave","kurt_wave", "entropy","forged_or_not")
head(data)
```

```
##   var_wave skew_wave kurt_wave  entropy forged_or_not
## 1  3.62160    8.6661   -2.8073 -0.44699             0
## 2  4.54590    8.1674   -2.4586 -1.46210             0
## 3  3.86600   -2.6383    1.9242  0.10645             0
## 4  3.45660    9.5228   -4.0112 -3.59440             0
## 5  0.32924   -4.4552    4.5718 -0.98880             0
## 6  4.36840    9.6718   -3.9606 -3.16250             0
```

This table shows the counts of fake and real bank notes

```
table(data$forged_or_not)
```

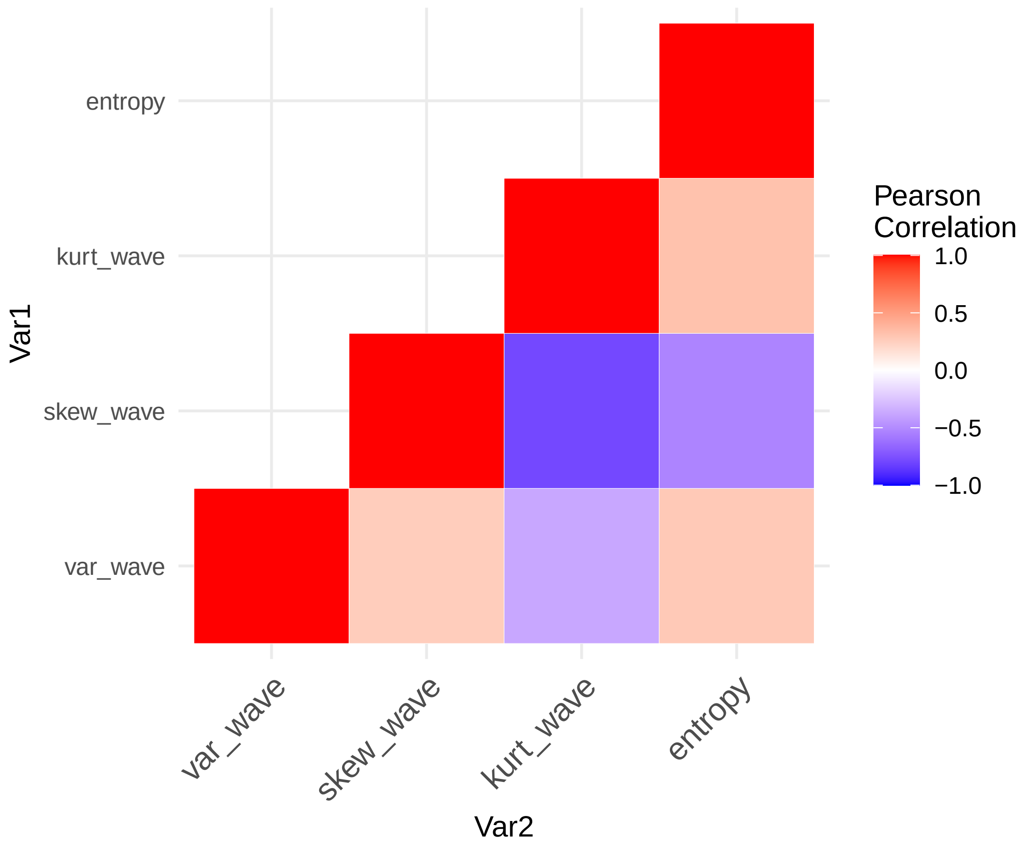
```
##
##   0   1
## 762 610
```

The data seems to be balanced with 762 real bank notes and 610 fake bank notes.

Plotting a heatmap to visually see the correlation between the variables

```
#For the heat map and correlation matrix
cor_data <- data[, 1:4]
cormat <- round(cor(cor_data),2)
melted_cormat <- melt(cormat)
# Get lower triangle of the correlation matrix
get_lower_tri<-function(cormat){
  cormat[upper.tri(cormat)] <- NA
  return(cormat)
}
# Get upper triangle of the correlation matrix
get_upper_tri <- function(cormat){
  cormat[lower.tri(cormat)]<- NA
  return(cormat)
}
upper_tri <- get_upper_tri(cormat)
# Melt the correlation matrix
```

```
melted_cormat <- melt(upper_tri, na.rm = TRUE)
# Heatmap
ggplot(data = melted_cormat, aes(Var2, Var1, fill = value))+
  geom_tile(color = "white")+
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                       midpoint = 0, limit = c(-1,1), space = "Lab",
                       name="Pearson\nCorrelation") +
  theme_minimal()+
  theme(axis.text.x = element_text(angle = 45, vjust = 1,
                                   size = 12, hjust = 1))+
  coord_fixed()
```



skew_wave and kurt_wave have strong negative correlation. All the other pairs are weakly correlated.

Splitting the data set

We now partition the data set into training and test data sets. We use the 'twinning' package since, twinning partitions the data based on their statistical properties(They try to build train and test data that are statistically similar).

```
set.seed(2468)
twin_indices = twin(data, r=5)
df_test = data[twin_indices, ]
df_train = data[-twin_indices, ]
```

Model Building and Validation

We now build the saturated model.

```
#Saturated model
fit0 = glm(forged_or_not ~ var_wave+ skew_wave+kurt_wave+entropy, data=df_train,
           family = binomial(link ="logit"))
```

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

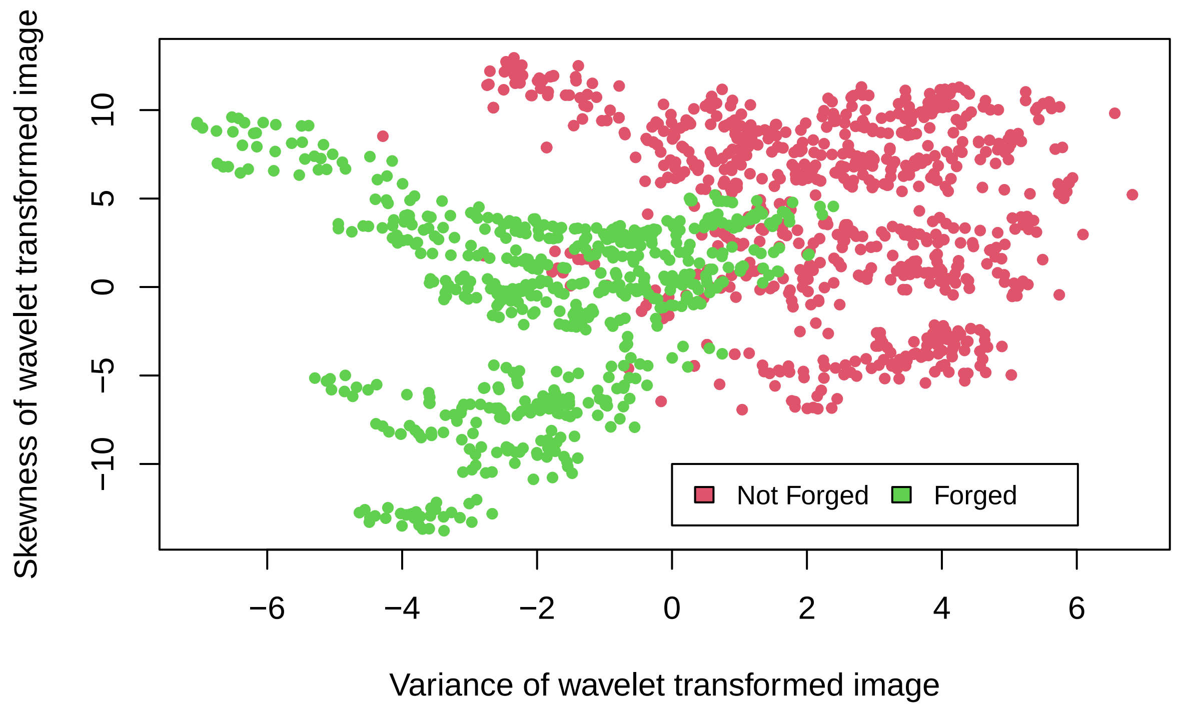
```
summary(fit0)
```

```
##
## Call:
## glm(formula = forged_or_not ~ var_wave + skew_wave + kurt_wave +
##     entropy, family = binomial(link = "logit"), data = df_train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.67779   0.00000   0.00000   0.00023   2.28466
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   7.2277     1.6838   4.293 1.77e-05 ***
## var_wave     -8.0031     1.9749  -4.052 5.07e-05 ***
## skew_wave    -4.2971     1.0314  -4.166 3.10e-05 ***
## kurt_wave    -5.3936     1.3166  -4.097 4.19e-05 ***
## entropy      -0.7079     0.3863  -1.832   0.0669 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1506.945  on 1096  degrees of freedom
## Residual deviance:   39.226  on 1092  degrees of freedom
## AIC: 49.226
##
## Number of Fisher Scoring iterations: 12
```

From the summary of this saturated model we can see that entropy is statistically insignificant. Along with that, from the heatmap we were able to see that kurt_wave is strongly correlated with skew_wave. So, we're going to drop these two variables from our model and build a new model.

Since we currently only have two variables in our feature space, we can visualize the feature space.

```
plot(data$var_wave,data$skew_wave,pch=20,col=(data$forged_or_not+2),
     xlab="Variance of wavelet transformed image",ylab="Skewness of wavelet transformed image")
legend(0,-10, legend=c("Not Forged", "Forged"),c(2,3), horiz=TRUE, cex=0.8)
```



We can see that the two groups have a peculiar shape. Even though there are a few problematic points in the midsection, we can clearly see that it is possible to classify them.

Now, we fit the updated model

```
fit1 = glm(forged_or_not ~ var_wave+ skew_wave, data=df_train,
           family = binomial(link ="logit"))
summary(fit1)
```

```
##
## Call:
## glm(formula = forged_or_not ~ var_wave + skew_wave, family = binomial(link = "logit"),
##     data = df_train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.54068  -0.33613  -0.05815   0.24357   2.58962
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.59003    0.12371    4.77 1.85e-06 ***
## var_wave    -1.10628    0.07189  -15.39  < 2e-16 ***
## skew_wave   -0.27688    0.02517  -11.00  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1506.94  on 1096  degrees of freedom
## Residual deviance:  576.41  on 1094  degrees of freedom
## AIC: 582.41
##
## Number of Fisher Scoring iterations: 6
```

We can get the predicted probabilities π's and try classifying the data points in the testing set. Before that we need to find the optimal threshold value from the using the model in the training set.

```
df_train$prob1 = predict(fit1,newdata = df_train,type="response")
op_thresh=0
op_accuracy=0
thresh_values = seq(0.10, 0.90, by=0.01)
for(i in thresh_values){
  new_pred_train= df_train$prob1
  new_pred_train<- ifelse(new_pred_train >i, 1, 0)
  missing_classerr <- mean(new_pred_train != df_train$forged_or_not)
  Accuracy= 1 - missing_classerr
  if(Accuracy>op_accuracy){
    op_thresh=i
    op_accuracy=Accuracy
  }
}
print(paste('Optimal Threshold value(upto two decimal points) =', op_thresh))
```

```
## [1] "Optimal Threshold value(upto two decimal points) = 0.5"
```

Now we apply the threshold value we found using the training data to classify the data points in the test set.

```
df_test$prob1 = predict(fit1,newdata = df_test,type="response")
df_test$prob1[df_test$prob1>op_thresh]=1
df_test$prob1[df_test$prob1<=op_thresh]=0
conf_tabl1 = table(predict = df_test$prob1,
                   actual = df_test$forged_or_not)
conf_tabl1
```

```
##        actual
## predict   0   1
##       0 139  17
##       1  13 106
```

The accuracy for fit1 is

```
acc1=sum(diag(conf_tabl1))/nrow(df_test)*100
acc1
```

```
## [1] 89.09091
```

Feature Engineering

The question we should ask is “Can we improve the classification?”. We saw that using a line to classify the points in our feature space might not be the best classifier that we can find. We can probably improve our classifier by adding quadratic terms. So, we update the model like this:

```
fit2 = update(fit1,.~.+I(var_wave^2)+I(skew_wave^2)+var_wave*skew_wave)
summary(fit2)
```

```
##
## Call:
## glm(formula = forged_or_not ~ var_wave + skew_wave + I(var_wave^2) +
##     I(skew_wave^2) + var_wave:skew_wave, family = binomial(link = "logit"),
##     data = df_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.4862  -0.2107  -0.0530   0.1516   2.2275
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)          1.54325    0.18918   8.158 3.41e-16 ***
## var_wave            -1.82143    0.15562 -11.704  < 2e-16 ***
## skew_wave           -0.05213    0.03630  -1.436    0.151
## I(var_wave^2)        0.05052    0.02675   1.888    0.059 .
## I(skew_wave^2)      -0.05462    0.00692  -7.894 2.93e-15 ***
## var_wave:skew_wave  0.13785    0.02339   5.893 3.79e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1506.94  on 1096  degrees of freedom
## Residual deviance:  417.59  on 1091  degrees of freedom
## AIC: 429.59
##
## Number of Fisher Scoring iterations: 7
```

We have to find the optimal threshold for this updated model using the training set. We just have to make a few tweaks to the code seen three code cells above.

```
## [1] "Optimal Threshold value(upto two decimal points) = 0.5"
```

Now we again apply the threshold value for fit2 we found using the training data to classify the data points in the test set.

```
df_test$prob2 = predict(fit2,newdata = df_test,type="response")
df_test$prob2[df_test$prob2>op_thresh]=1
df_test$prob2[df_test$prob2<=op_thresh]=0
conf_tabl2 = table(predict = df_test$prob2,
                   actual = df_test$forged_or_not)
conf_tabl2
```

```
##         actual
## predict   0   1
##       0 141  13
##       1  11 110
```

The accuracy of fit2

```
acc2=sum(diag(conf_tabl2))/nrow(df_test)*100
acc2
```

```
## [1] 91.27273
```

We see that the accuracy has increased from 89.1% to 91.27% upon adding quadratic terms in our classifier.

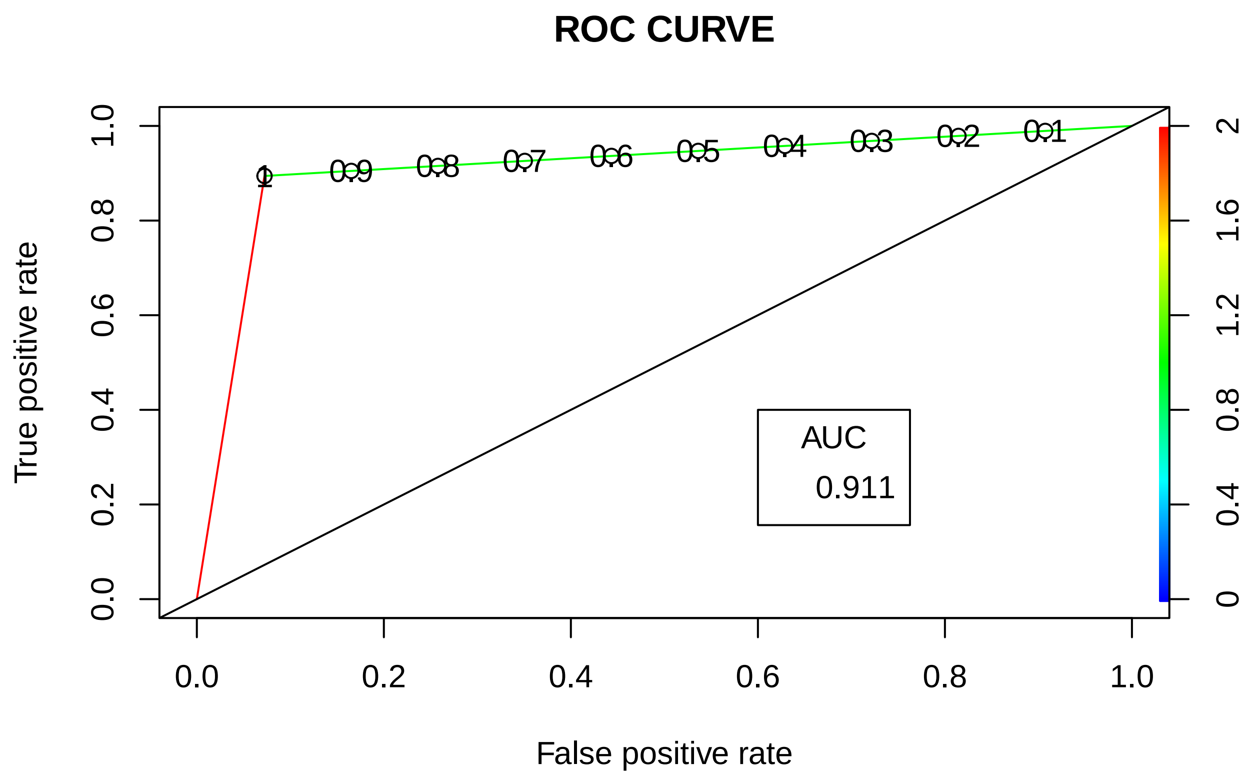
Now if we plot the ROC Curve to see how good our model predicts on the test set,

```
ROCPred <- prediction(df_test$prob2, df_test$forged_or_not)
ROCPer <- performance(ROCPred, measure = "tpr",
x.measure = "fpr")
auc <- performance(ROCPred, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
## [1] 0.9109703
```

We see that Area Under the Curve(AUC) is 0.911

```
plot(ROCPer, colorize = TRUE,
print.cutoffs.at = seq(0.1, by = 0.1),
main = "ROC CURVE")
abline(a = 0, b = 1)
auc <- round(auc, 4)
legend(.6, .4, auc, title = "AUC", cex = 1)
```



AUC of is a very high value, so fit2 is a very good classifier.

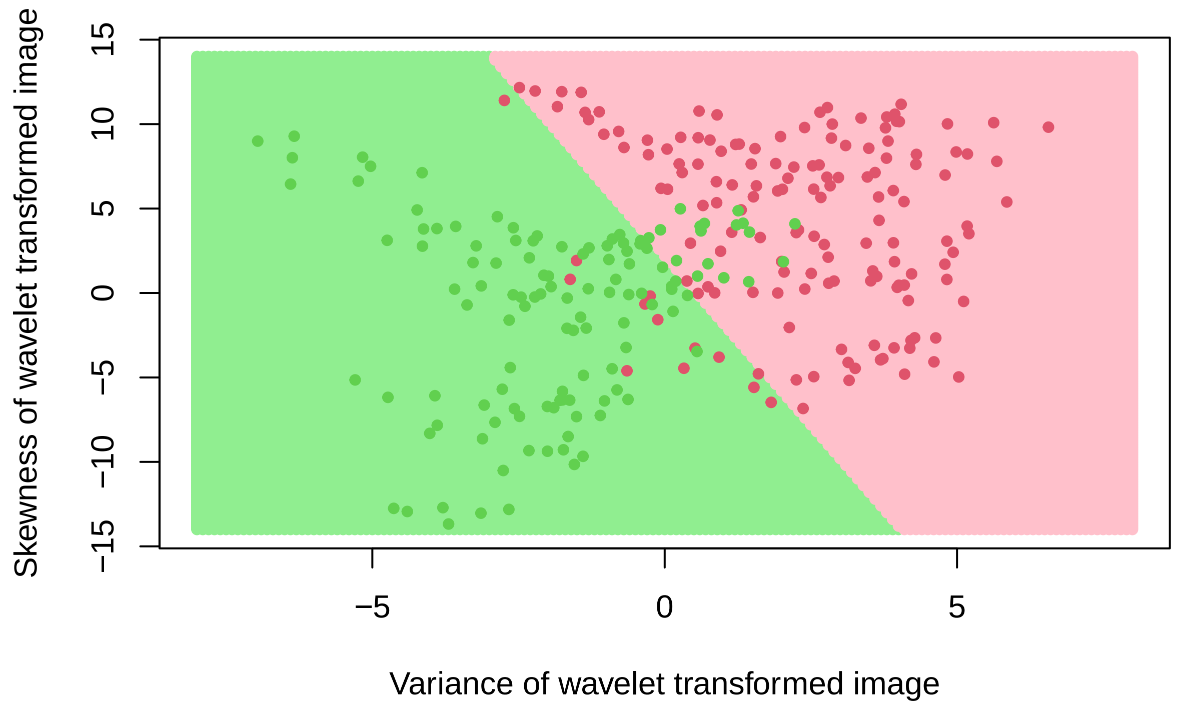
Visual Comparison of the two models

The accuracy went up as we added quadratic terms to our model, let's try finding out why that happened. Let's try doing that graphically.

```r
#visualization to understand the effect of feature engineering
test_data = dum_var = data.frame(matrix(NA,nrow=1,ncol=2))
colnames(test_data)= colnames(dum_var)=c("var_wave","skew_wave")

var_wave = seq(-8,8, by=0.1)
skew_wave = seq(-14,14, by=0.1)

for(i in var_wave){
  for(j in skew_wave){
    dum_var=c(i,j)
    test_data = rbind.data.frame(test_data, dum_var)
  }
}

test_data=na.omit(test_data)
```

This is for the classifier that only uses linear terms of the features.

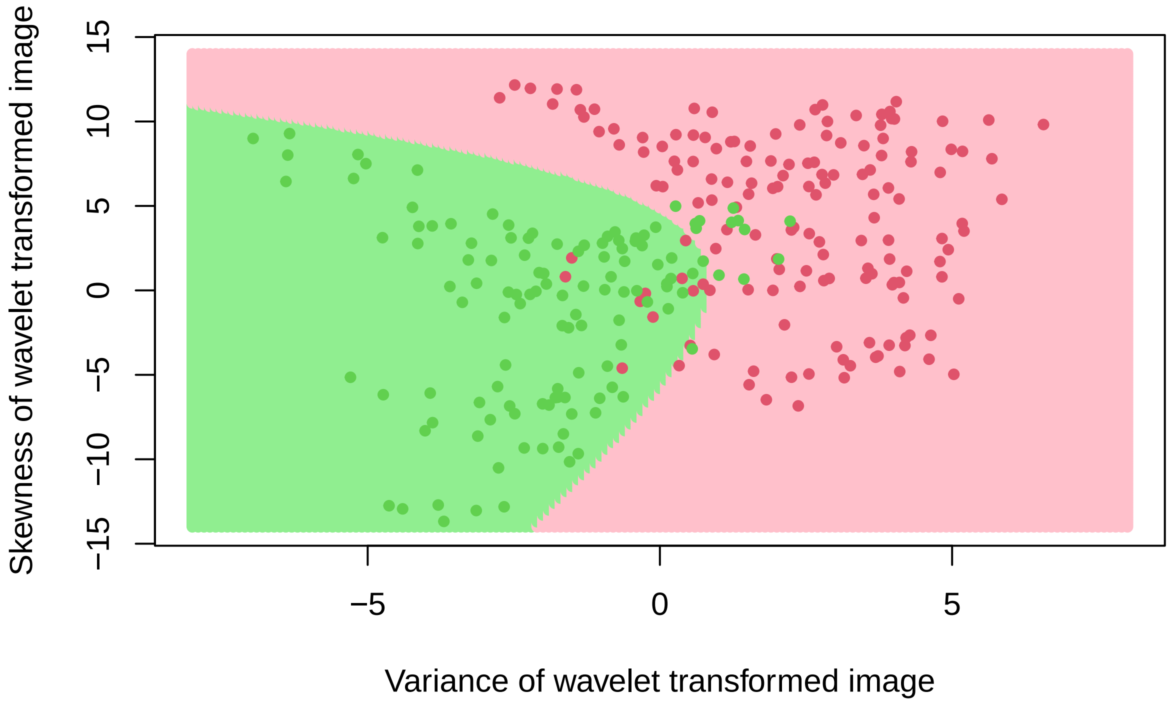
```
test_data$prob1 = predict(fit1,newdata = test_data,type="response")
test_data$pred1=NA
test_data$pred1[test_data$prob1>0.5]="lightgreen"
test_data$pred1[test_data$prob1<=0.5]="pink"

plot(test_data$var_wave,test_data$skew_wave,
     xlim =c(-8,8),ylim=c(-14,14),
     pch=20,col=test_data$pred1,
     xlab="Variance of wavelet transformed image",
     ylab="Skewness of wavelet transformed image")

points(df_test$var_wave,df_test$skew_wave,
       pch=20,col=(df_test$forged_or_not+2))
```



This is for the classifier that uses quadratic terms of the features.

```
test_data$prob2 = predict(fit2,newdata = test_data,type="response")
test_data$pred2=NA
test_data$pred2[test_data$prob2>0.5]="lightgreen"
test_data$pred2[test_data$prob2<=0.5]="pink"

plot(test_data$var_wave,test_data$skew_wave,
     xlim =c(-8,8),ylim=c(-14,14),
     pch=20,col=test_data$pred2,
     xlab="Variance of wavelet transformed image",
     ylab="Skewness of wavelet transformed image")
```

```
points(df_test$var_wave,df_test$skew_wave,
       pch=20,col=(df_test$forged_or_not+2))
```



We can clearly see that the one with the quadratic terms has fewer missclassified points.