SVM Classification

Aditi Chaudhari and Abigail Solomon

SVM Classification

Generally, in Classification technique, input data is labeled in accordance with the historical data samples, and then manually trained to identify the class of the new given data.In SVM Classification, given the transactions made by the credit cards as predictors, support vectors will represent the coordinate representation of individual observation, that we utilize for segregating two classes, whether the credit card transaction is fraudulent or not.

Load necessary libraries

```
library(e1071)
```

Import the Data set

Source of the Data Set

Credit Card Transactions data set: 'creditcard'dataset (https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud)

Read data

We use the read.csv() function to read a csv. Using the dim() structure function, We will see that the data set contain transactions made by credit cards, and it has 284807 observations of 31 variables along with their names.

```
#Read data
df <- read.csv("creditcard.csv")
dim(df)
```

```
## [1] 284807     31
```

```
names(df)
```

```
##  [1] "Time"   "V1"     "V2"     "V3"     "V4"     "V5"     "V6"     "V7"
##  [9] "V8"     "V9"     "V10"    "V11"    "V12"    "V13"    "V14"    "V15"
## [17] "V16"    "V17"    "V18"    "V19"    "V20"    "V21"    "V22"    "V23"
## [25] "V24"    "V25"    "V26"    "V27"    "V28"    "Amount" "Class"
```

Data Preprocessing

Data processing is converting train raw data sets into meaningful sets that are usable.We will be examining the specific types and steps of data cleaning and later the scaling before analyzing the data.

Data cleaning

In order to make our data set for machine learning more meaningful, we may need to fix or remove missing or unwanted, data instances which may not help to solve the problem, from the data set. The sapply() function gives the number of missing values in each column, in this case we don't have any NA's.

```
sapply(df, function(x) sum(is.na(x)==TRUE))
```

```
##   Time     V1     V2     V3     V4     V5     V6     V7     V8     V9    V10
##      0      0      0      0      0      0      0      0      0      0      0
##    V11    V12    V13    V14    V15    V16    V17    V18    V19    V20    V21
##      0      0      0      0      0      0      0      0      0      0      0
##    V22    V23    V24    V25    V26    V27    V28 Amount  Class
##      0      0      0      0      0      0      0      0      0
```

Data Sampling

Large data sets consume a lot of memory and took ages to run, in our case we are going to take only the first 20,000 observations,but first we will use the unique function that repetitions will be removed, still 283726 is a big data. Since we don't have NA 's, we will randomly remove the rows after the 20,000th.

```
df <- unique(df)
dim(df)
```

```
## [1] 283726     31
```

```
names(df)
```

```
##  [1] "Time"   "V1"     "V2"     "V3"     "V4"     "V5"     "V6"     "V7"
##  [9] "V8"     "V9"     "V10"    "V11"    "V12"    "V13"    "V14"    "V15"
## [17] "V16"    "V17"    "V18"    "V19"    "V20"    "V21"    "V22"    "V23"
## [25] "V24"    "V25"    "V26"    "V27"    "V28"    "Amount" "Class"
```

```
new_df <- df[-c(20001:283726), ]
str(new_df)
```

```
## 'data.frame':    20000 obs. of  31 variables:
##  $ Time  : num  0 0 1 1 2 2 4 7 7 9 ...
##  $ V1    : num  -1.36 1.192 -1.358 -0.966 -1.158 ...
##  $ V2    : num  -0.0728 0.2662 -1.3402 -0.1852 0.8777 ...
##  $ V3    : num  2.536 0.166 1.773 1.793 1.549 ...
##  $ V4    : num  1.378 0.448 0.38 -0.863 0.403 ...
##  $ V5    : num  -0.3383 0.06 -0.5032 -0.0103 -0.4072 ...
##  $ V6    : num  0.4624 -0.0824 1.8005 1.2472 0.0959 ...
##  $ V7    : num  0.2396 -0.0788 0.7915 0.2376 0.5929 ...
##  $ V8    : num  0.0987 0.0851 0.2477 0.3774 -0.2705 ...
##  $ V9    : num  0.364 -0.255 -1.515 -1.387 0.818 ...
##  $ V10   : num  0.0908 -0.167 0.2076 -0.055 0.7531 ...
##  $ V11   : num  -0.552 1.613 0.625 -0.226 -0.823 ...
##  $ V12   : num  -0.6178 1.0652 0.0661 0.1782 0.5382 ...
##  $ V13   : num  -0.991 0.489 0.717 0.508 1.346 ...
##  $ V14   : num  -0.311 -0.144 -0.166 -0.288 -1.12 ...
##  $ V15   : num  1.468 0.636 2.346 -0.631 0.175 ...
##  $ V16   : num  -0.47 0.464 -2.89 -1.06 -0.451 ...
##  $ V17   : num  0.208 -0.115 1.11 -0.684 -0.237 ...
##  $ V18   : num  0.0258 -0.1834 -0.1214 1.9658 -0.0382 ...
##  $ V19   : num  0.404 -0.146 -2.262 -1.233 0.803 ...
##  $ V20   : num  0.2514 -0.0691 0.525 -0.208 0.4085 ...
##  $ V21   : num  -0.01831 -0.22578 0.248 -0.1083 -0.00943 ...
##  $ V22   : num  0.27784 -0.63867 0.77168 0.00527 0.79828 ...
##  $ V23   : num  -0.11 0.101 0.909 -0.19 -0.137 ...
##  $ V24   : num  0.0669 -0.3398 -0.6893 -1.1756 0.1413 ...
##  $ V25   : num  0.129 0.167 -0.328 0.647 -0.206 ...
##  $ V26   : num  -0.189 0.126 -0.139 -0.222 0.502 ...
##  $ V27   : num  0.13356 -0.00898 -0.05535 0.06272 0.21942 ...
##  $ V28   : num  -0.0211 0.0147 -0.0598 0.0615 0.2152 ...
##  $ Amount: num  149.62 2.69 378.66 123.5 69.99 ...
##  $ Class : int  0 0 0 0 0 0 0 0 0 0 ...
```

Data Split

We are going to split our data set into training, validation and tests sets. 60% of our data will be attributed to the train data set, 20% will be attributed to the validation data, and the rest 20% will be attributed to the test data. We will then find the dimensions of the data frame using the dim() function.

```
set.seed(1234)
groups <- c(train=.6, test=.2, validate=.2)
i <- sample(cut(1:nrow(new_df),
nrow(new_df)*cumsum(c(0,groups)), labels=names(groups)))
train <- new_df[i=="train",]
test <- new_df[i=="test",]
vald <- new_df[i=="validate",]
dim(new_df)
```

```
## [1] 20000    31
```

Data Exploration

Data exploration helps us to gain insight into the raw train data and findings of R built-in functions.We will print the first and last six rows of transactions using the head() and tail() functions respectively.The summary () function applied on the Amount vector, calculates summary statistics for each of them, it prints the Minimum value, the 1st quartile's value (25th percentile), the median value, the 3rd quartile's value (75th percentile) and the maximum value.

The first six transactions

```
head(train)
```

	Time <dbl>	V1 <dbl>	V2 <dbl>	V3 <dbl>	V4 <dbl>	V5 <dbl>	V6 <dbl>	V7 <dbl>
1	0	-1.3598071	-0.07278117	2.5363467	1.3781552	-0.33832077	0.46238778	0.23959855
2	0	1.1918571	0.26615071	0.1664801	0.4481541	0.06001765	-0.08236081	-0.07880298
3	1	-1.3583541	-1.34016307	1.7732093	0.3797796	-0.50319813	1.80049938	0.79146096
4	1	-0.9662717	-0.18522601	1.7929933	-0.8632913	-0.01030888	1.24720317	0.23760894
5	2	-1.1582331	0.87773675	1.5487178	0.4030339	-0.40719338	0.09592146	0.59294075
6	2	-0.4259659	0.96052304	1.1411093	-0.1682521	0.42098688	-0.02972755	0.47620095

6 rows | 1-9 of 32 columns

The last six transactions

```
tail(train)
```

	Time	V1	V2	V3	V4	V5	V6	V7
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
20058	30756	1.0363535	0.1636831	0.6760946	2.5584767	-0.08399691	0.39530840	-0.001411434
20059	30757	-1.1228884	0.9671246	1.4819611	1.2180410	1.61782853	-0.07645927	0.909090144
20060	30757	-0.4517422	0.5465346	0.7184858	0.8112181	0.86372428	1.65002255	1.191292394
20064	30759	-0.6557226	1.0247718	0.3228922	1.4597150	2.87549909	4.17833014	0.031660607
20065	30760	-0.5350055	1.7036076	0.3046770	1.1513854	0.17376943	-1.10207625	0.528181003
20069	30762	-9.6155288	4.8909165	-7.5072154	1.3770384	-5.75901587	-1.03335745	-3.554575092

6 rows | 1-9 of 32 columns

Summary of the total Amount

```
summary(train[c('Amount')])
```

```
##      Amount
##  Min.   :   0.00
##  1st Qu.:   5.49
##  Median :  16.31
##  Mean   :  71.69
##  3rd Qu.:  60.00
##  Max.   :7879.42
```

Visual Data Exploration

Data visualization present train data contents in graphical or picture format, enables us to grasp and understand analytics in an easier manner and be able to communicate what has been learned about the data to others, it is also optically entertaining.The Amount and the Class of the data set are plotted using Histogram, Scatter plot and Kernel density plot.

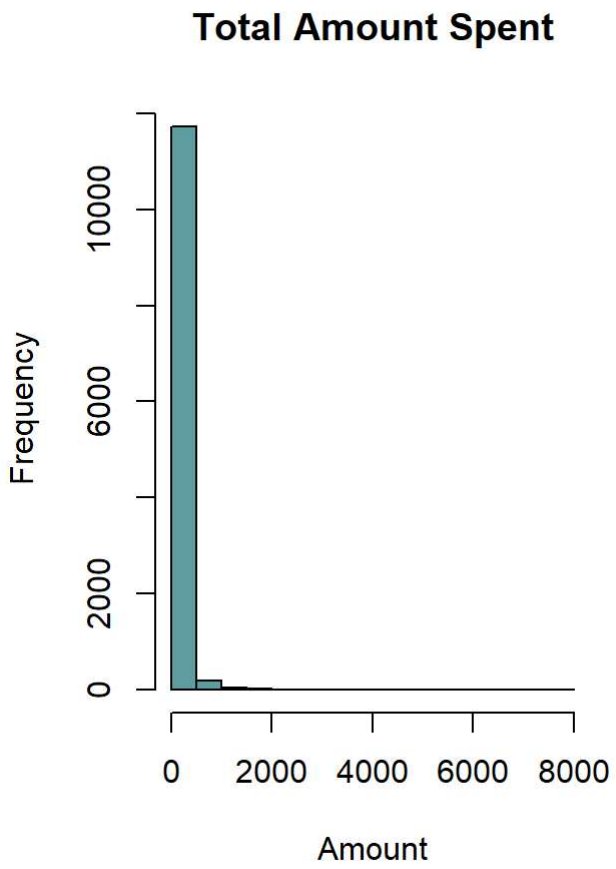
Histogram

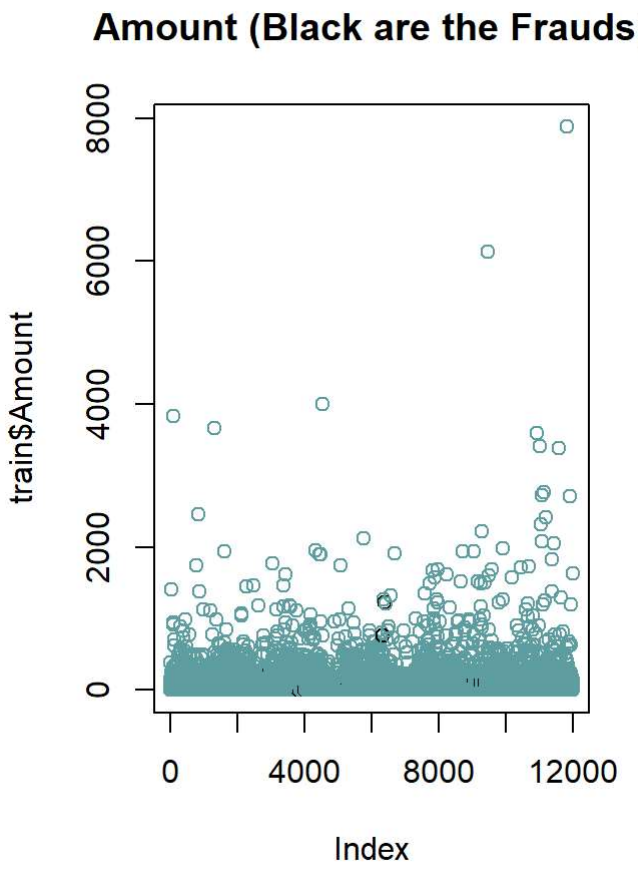
The Histogram graph displays the frequency of the x values, in our case it shows that small amount is spent most of the time, which is similar to the mean(Amount), which is 71.69 The plot graph similarly displays that small amount, less than 2000 are the dominant ones, the black spots show the fraud ones,tiny amount, from the findings of the Class summary, the fraud are only 0.004% of all the transaction.

```
#copy original settings
opar <- par()
#set up 1x2 grid
par(mfrow=c(1,2))

hist(train$Amount, col="cadetblue", main="Total Amount Spent", xlab="Amount")

plot(train$Amount, col=c("cadetblue","black")[train$Class + 1], main="Amount (Black are the Frauds)")
```



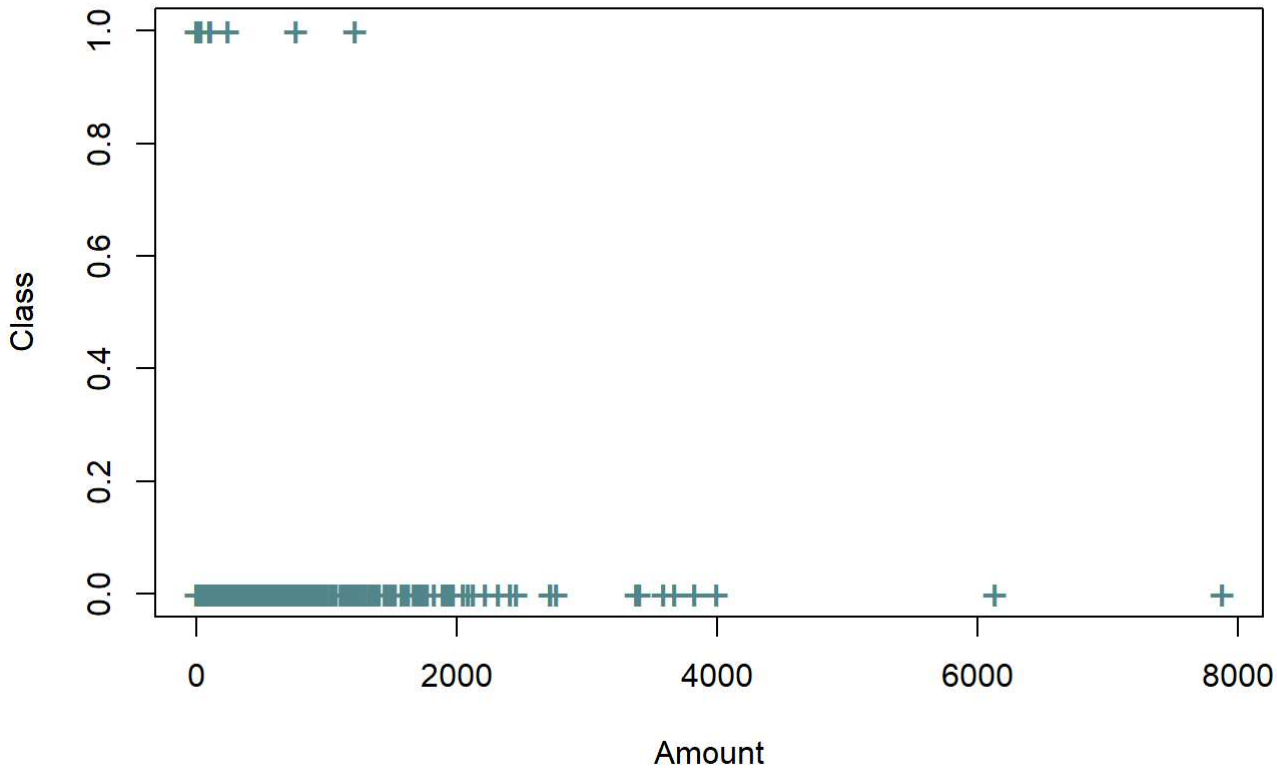



```
par(opar) # restore parameter settings
```

Scatter Plot

In scatter plots, we can see the Amount versus Class similar to the results of the summary of the Class, small transactions are fraud, but the majority not.It also shows that the small amount, less than 2000 are crowded, many in number.

```
plot(train$Amount, train$Class, pch='+', cex=1.5, col="cadetblue4", xlab="Amount", ylab="Class")
```

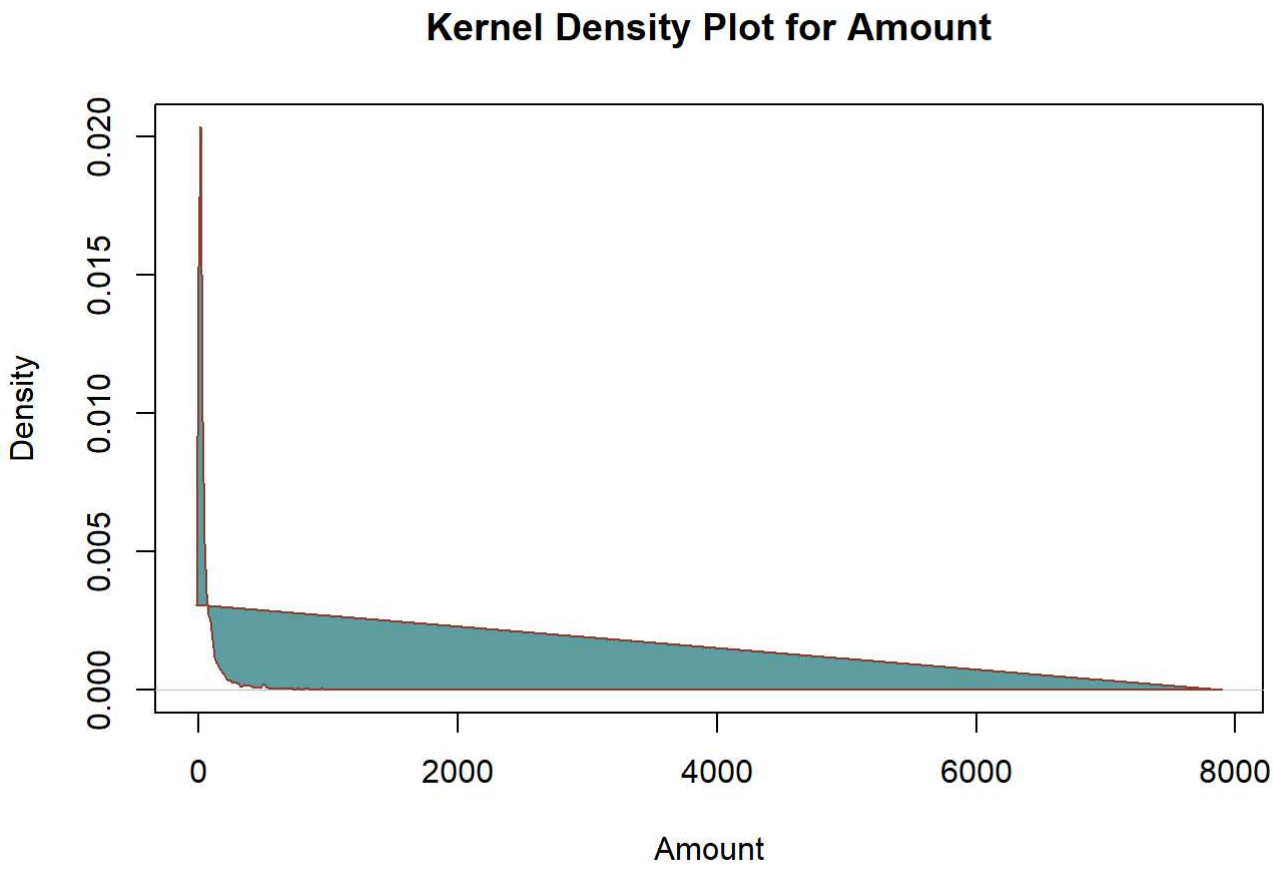


Kernel Density Plot

A kernel density plot is similar to a histogram, but it displays the distribution of values in a data set using one continuous curve.It is better at displaying the shape of a distribution since it isn't affected by the number of bins used in the histogram.It displays that the small amount,less than 2000 are the dense ones just like the histogram.

```
#The Plot graph shows that

d <- density(train$Amount)
plot(d, main="Kernel Density Plot for Amount", xlab="Amount")
polygon(d, col="cadetblue", border="coral4")
```



Summary of Class

The summary() function returns different values according to the given parameter, since the class is assigned as factor,the summary of the Class vector retrieve the amount of each level, that is the Boolean values namely “0” and “1”. The “0” means the transaction is not fraudulent, where as “1” shows that the transaction is fraudulent. We can see that 11,952 are not frauds, only 48 transactions, which are 0.004% of 12,000 transactions are frauds.

```
train$Class <- as.factor(train$Class)
summary(train[c('Class')])
```

```
##  Class
##  0:11952
##  1:   48
```

Scaling

The scale() function is simply standardization of the data, it is useful when you have multiple variables across different scales. Let's apply the function to the Amount Column, the data will be structured according to the specified range.We remove the Time column since it's not an essential data,let's print the head() once more,we can see here that the data is scaled and the column Time is not included since we omitted it.

```
train$Amount=scale(train$Amount)
# Remove Time column
train$Time <- NULL
head(train)
```

	V1 <dbl>	V2 <dbl>	V3 <dbl>	V4 <dbl>	V5 <dbl>	V6 <dbl>	V7 <dbl>	V8 <dbl>
1	-1.3598071	-0.07278117	2.5363467	1.3781552	-0.33832077	0.46238778	0.23959855	0.09869790
2	1.1918571	0.26615071	0.1664801	0.4481541	0.06001765	-0.08236081	-0.07880298	0.08510165
3	-1.3583541	-1.34016307	1.7732093	0.3797796	-0.50319813	1.80049938	0.79146096	0.24767579
4	-0.9662717	-0.18522601	1.7929933	-0.8632913	-0.01030888	1.24720317	0.23760894	0.37743587
5	-1.1582331	0.87773675	1.5487178	0.4030339	-0.40719338	0.09592146	0.59294075	-0.27053268
6	-0.4259659	0.96052304	1.1411093	-0.1682521	0.42098688	-0.02972755	0.47620095	0.26031433

6 rows | 1-9 of 31 columns

Data Modeling

We are going to build the linear SVM classification, the Polynomial SVM,and the Radial Kernels SVM and see the accuracy of each model.

Linear Logistic Regression model

Summary of the model:

```
glm1 <- glm(Class ~ data.matrix(Amount), data=train, family="binomial")
summary(glm1)
```

```
##
## Call:
## glm(formula = Class ~ data.matrix(Amount), family = "binomial",
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.1184  -0.0895  -0.0893  -0.0893   3.3247
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -5.51757    0.14464 -38.146   <2e-16 ***
## data.matrix(Amount)  0.01492    0.13067   0.114    0.909
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 625.87  on 11999  degrees of freedom
## Residual deviance: 625.86  on 11998  degrees of freedom
## AIC: 629.86
##
## Number of Fisher Scoring iterations: 8
```

Evaluate on the test set

The model has 95% accuracy.We have 1 TP value, that 1 fraud transactions and 3832 TN, not fraud transactions.

```
probs <- predict(glm1, newdata=test, type="response")
pred <- ifelse(probs>0.5, 1, 0)
acc <- mean(pred==test$Class)
print(paste("accuracy = ", acc))
```

```
## [1] "accuracy =  0.95825"
```

```
table(pred, test$Class)
```

```
##
## pred    0    1
##    0 3832   15
##    1  152    1
```

Linear SVM model

Let us build an SVM1 model on the train set using cost=10 and kernel="linear".

```
svm1 <- svm(Class~., data=train, kernel="linear", cost=10, scale=TRUE)
summary(svm1)
```

```
##
## Call:
## svm(formula = Class ~ ., data = train, kernel = "linear", cost = 10,
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  56
##
##  ( 36 20 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Evaluate the linear svm (SVM1)

Now we have a model, we can predict the value of the new data set, which is our test data set by giving inputs to our model.The table provides True positive(TP) and True negative (TN) values in the diagonal, so we have 8 TP value, that 8 fraud transactions and 3983 TN, not fraud transactions.The model has 99% accuracy.Fraud transactions are 0.002% of 3983. We remember that in the raw data, we had 0.004% of 12,000 transactions are frauds, quite similar.

```
svm_pred1 <- predict(svm1, newdata=test)
table(svm_pred1, test$Class)
```

```
##
## svm_pred1    0    1
##         0 3983    8
##         1    1    8
```

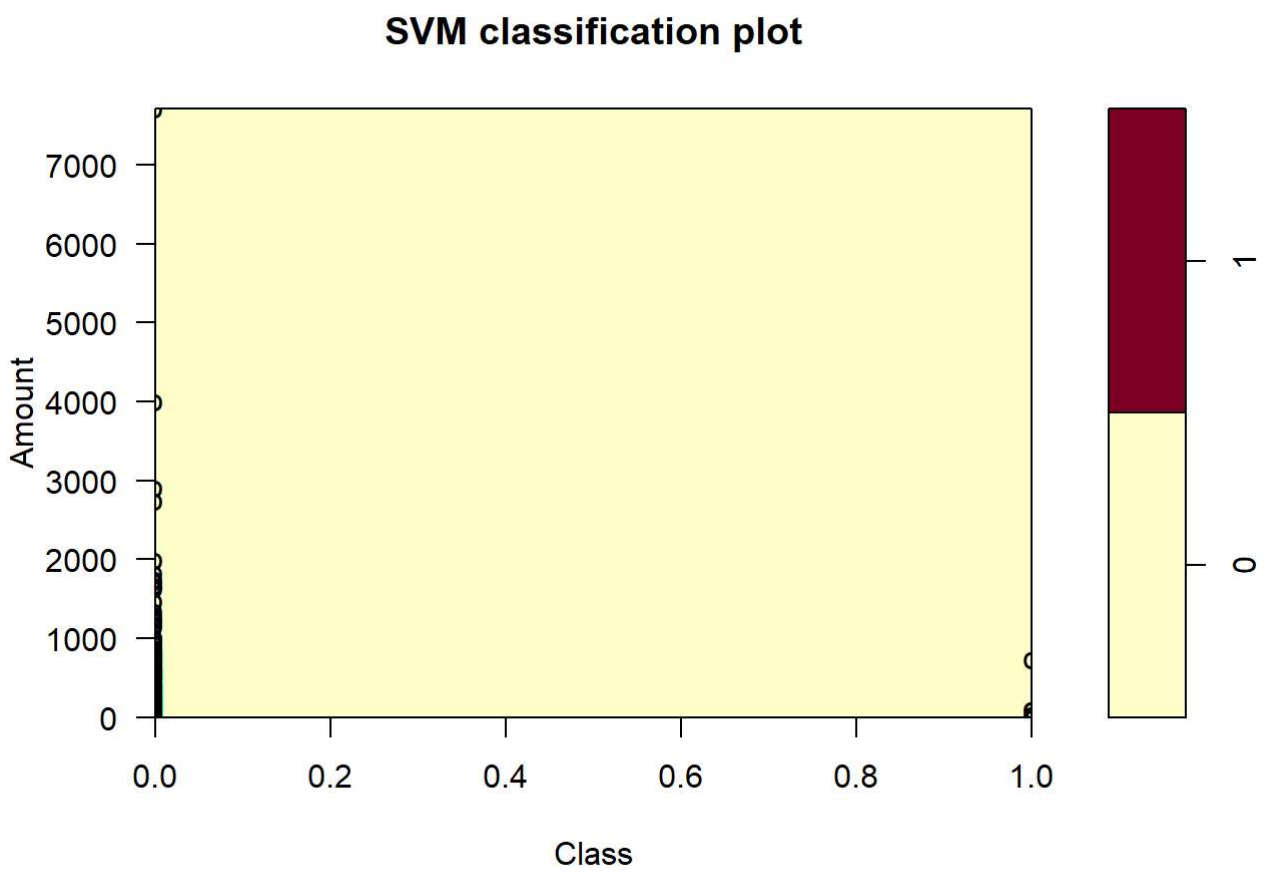
```
mean(svm_pred1==test$Class)
```

```
## [1] 0.99775
```

Linear svm Plot

We almost don’t see the brown color for the frauds,since they have tiny value. The data is unbalanced data collected from two day’s transactions.
Plot the Support Vectors

```
plot(svm1,test, Amount~Class)
```



Linear svm Tuning

The cost parameter determines how much slack variables will be allowed.Experiment with various cost values to get the best model.The hyperparameters are tuned on the validation set to not over fit data and not against good principles by letting the algorithm see test data.Larger C have larger margins, smaller C, move the model toward lower bias, higher variance. The summary of tune_svm1 tells us the best cost is 10.The next syntax will use the best model value to make predictions on the test data.The least error has the 10 cost value.Best performance: 0.002506109

```
set.seed(1234)
tune_svm1 <- tune(svm, Class~., data=vald, kernel="linear",
ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune_svm1)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.002506109
##
## - Detailed performance results:
##    cost       error  dispersion
## 1 1e-03 0.003585853 0.002129054
## 2 1e-02 0.002535909 0.001738260
## 3 1e-01 0.002508968 0.001736417
## 4 1e+00 0.002506838 0.001735125
## 5 5e+00 0.002506420 0.001734804
## 6 1e+01 0.002506109 0.001734602
## 7 1e+02 0.002506827 0.001726301
```

```
best_model <- tune_svm1$best.model
summary(best_model)
```

```
##
## Call:
## best.tune(method = svm, train.x = Class ~ ., data = vald, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.03333333
##     epsilon:  0.1
##
##
## Number of Support Vectors:  67
```

```
pred <- predict(best_model, newdata=test)
acc_svm2 <- mean(pred==test$Class)
```

Polynomial SVM model

Let us build an SVM2 model on the train set using cost=10 and kernel=“Polynomial”.

```
svm2 <- svm(Class~., data=train, kernel="polynomial", cost=10, scale=TRUE)
summary(svm2)
```

```
##
## Call:
## svm(formula = Class ~ ., data = train, kernel = "polynomial", cost = 10,
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  68
##
##  ( 55 13 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Evaluate the polynomial svm (SVM2)

Let us predict the value of the new data set with the model, SVM2, which is our test data set by giving inputs to our model.The table provides True positive(TP) and True negative (TN) values in the diagonal, so we have 12 TP value, that 12 fraud transactions and 3978 TN, not fraud transactions.The model has 99% accuracy,the same accurate as SVM linear.

```
svm_pred2 <- predict(svm2, newdata=test)
table(svm_pred2, test$Class)
```

```
##
## svm_pred2    0    1
##         0 3978    4
##         1    6   12
```

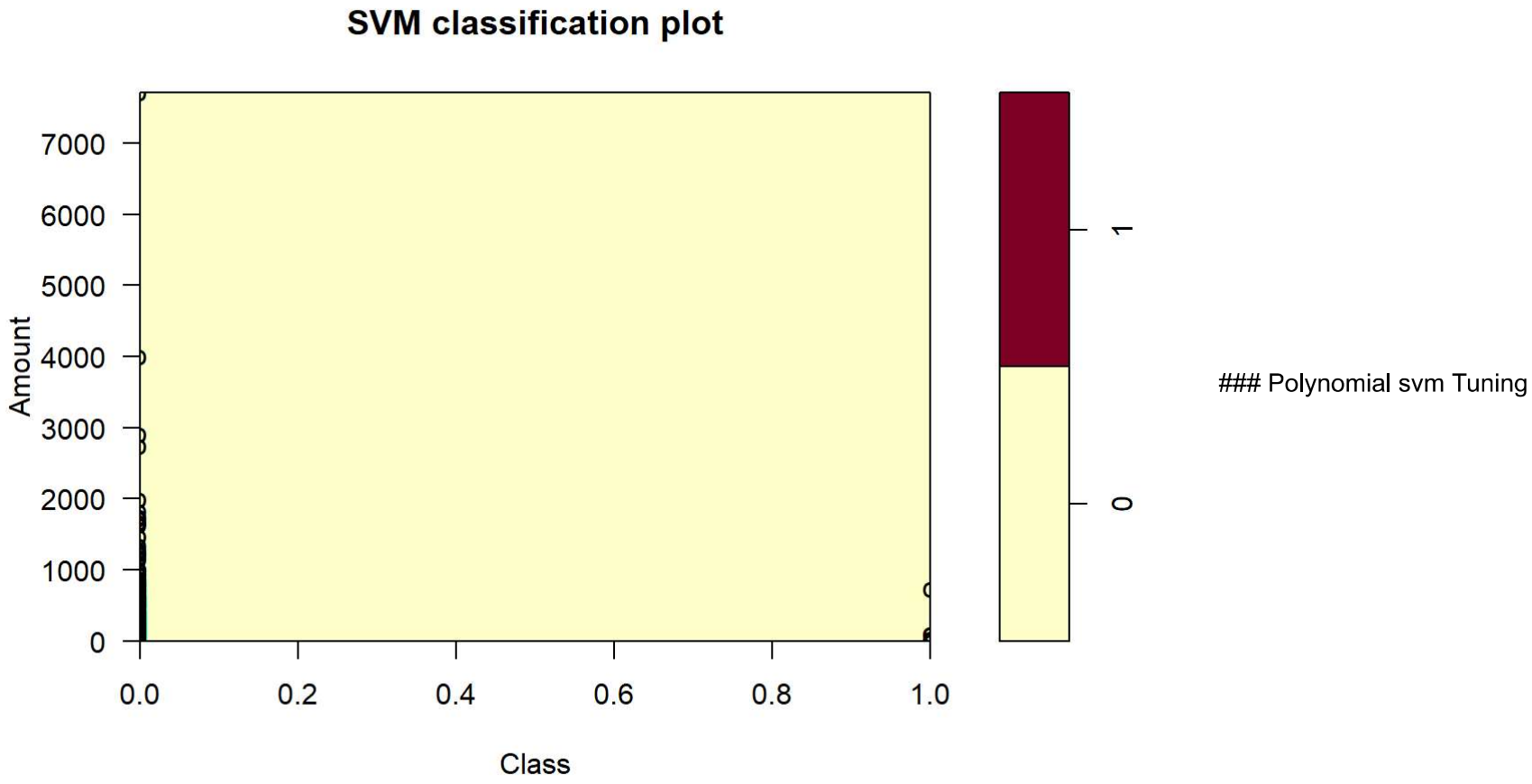
```
mean(svm_pred2==test$Class)
```

```
## [1] 0.9975
```

Polynomial svm Plot

Plot the Support Vectors

```
plot(svm2,test, Amount~Class)
```



Polynomial svm Tuning

The cost parameter determines how much slack variables will be allowed.Experiment with various cost values to get the best model.The hyperparameters are tuned on the validation set to not over fit data and not against good principles by letting the algorithm see test data.Larger C have larger margins, smaller C, move the model toward lower bias, higher variance. The summary of tune_svm1 tells us the best cost is 0.1.The next syntax will use the best model value to make predictions on the test data.The least error has the 0.1 cost value.Best performance: 0.00174921

```
set.seed(1234)
tune_svm2 <- tune(svm, Class~., data=vald, kernel="polynomial",
ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune_svm2)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  0.001
##
## - best performance: 0.003553607
##
## - Detailed performance results:
##    cost       error dispersion
## 1 1e-03 0.003553607 0.00228555
## 2 1e-02 0.007028990 0.01145713
## 3 1e-01 0.016330418 0.03970911
## 4 1e+00 0.026980783 0.04721266
## 5 5e+00 0.137411227 0.30292071
## 6 1e+01 0.206353047 0.50583105
## 7 1e+02 0.180216892 0.42468459
```

```
best_model <- tune_svm2$best.model
summary(best_model)
```

```
##
## Call:
## best.tune(method = svm, train.x = Class ~ ., data = vald, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  polynomial
##        cost:  0.001
##      degree:  3
##       gamma:  0.03333333
##      coef.0:  0
##     epsilon:  0.1
##
##
## Number of Support Vectors:  53
```

```
pred <- predict(best_model, newdata=test)
acc_svm3 <- mean(pred==test$Class)
```

Radial Kernel SVM model

Let us build SVM3 model on the train set using cost=1 and kernel="radial" and predict the value of the new data set.

```
svm3 <- svm(Class~., data=train, kernel="radial",
cost=1, gamma=1, scale=FALSE)
summary(svm3)
```

```
##
## Call:
## svm(formula = Class ~ ., data = train, kernel = "radial", cost = 1,
##     gamma = 1, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  9087
##
##  ( 9039 48 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Evaluate the radial kernel svm (SVM3)

Let us predict the value of the new data set with the model, SVM3, which is our test data set by giving inputs to our model.The table provides 0 TP value, that 0 fraud transactions and 3984 TN, not fraud transactions.The model has 99% accuracy, just as accurate as the polynomial svm and linear svm

```
svm_pred3 <- predict(svm3, newdata=test)
table(svm_pred3, test$Class)
```

```
##
## svm_pred3    0    1
##         0 3984   16
##         1    0    0
```

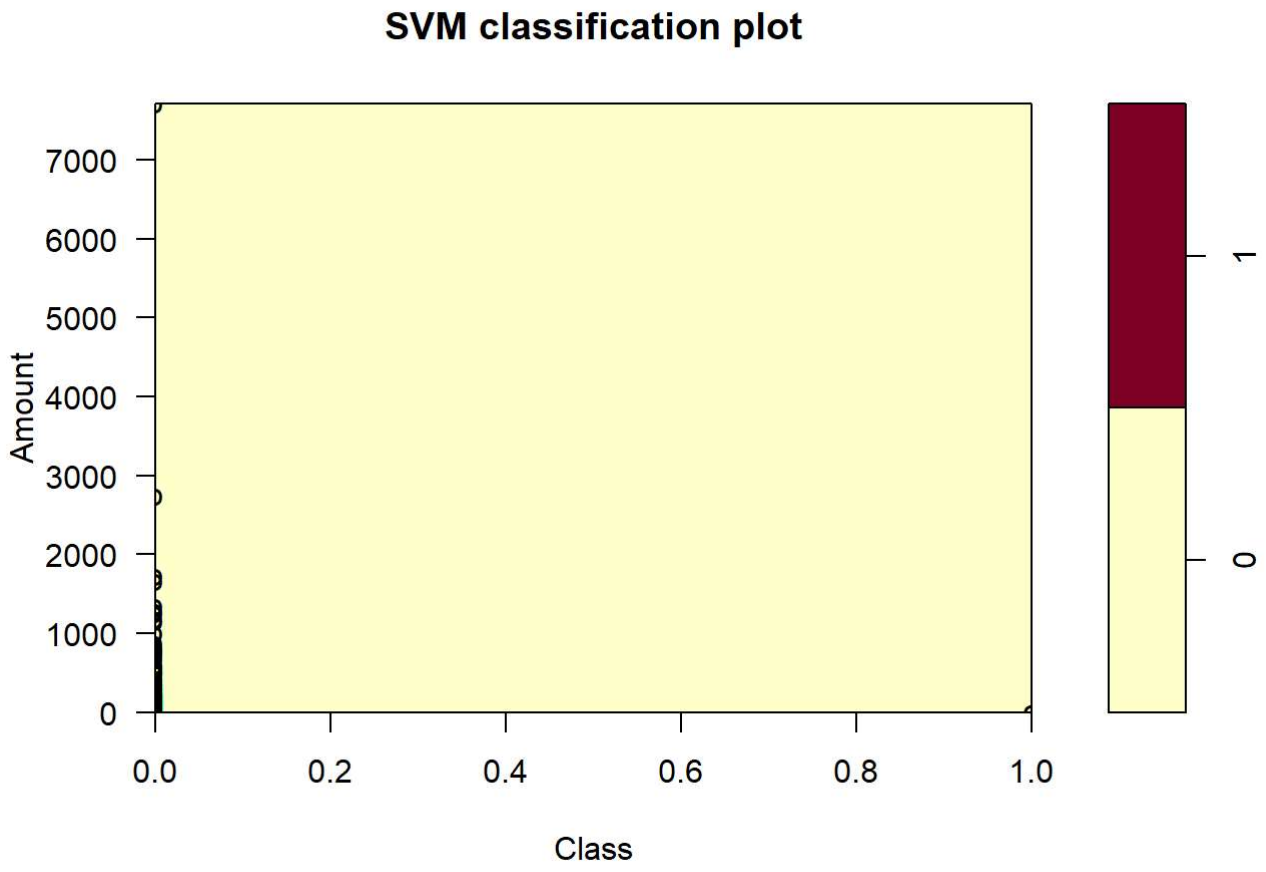
```
mean(svm_pred3==test$Class)
```

```
## [1] 0.996
```

SVM3 Classification Plot

Plot the Support Vectors

```
plot(svm3,test, Amount~Class)
```



SVM3 Tuning

The gamma hyperparameter is tuned on validation data,larger gamma can over fit and move the model toward high variance, and lower gamma can under fit, leading the model with high bias.The summary of tune_svm3 tells us the best cost is 100 and gamma is 0.5.The next syntax will use the best model value to make predictions on the test data.The least error has the 100 cost value.Best performance: 0.003744152

```
set.seed(1234)
tune_svm3 <- tune(svm, Class~., data=vald, kernel="radial",
ranges=list(cost=c(0.1,1,10,100,1000),
gamma=c(0.5,1,2,3,4)))
summary(tune_svm3)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100   0.5
##
## - best performance: 0.003744152
##
## - Detailed performance results:
##     cost gamma       error  dispersion
## 1  1e-01   0.5 0.005206889 0.003547907
## 2  1e+00   0.5 0.005021094 0.003356840
## 3  1e+01   0.5 0.003878406 0.002439860
## 4  1e+02   0.5 0.003744152 0.002375065
## 5  1e+03   0.5 0.003744152 0.002375065
## 6  1e-01   1.0 0.005214010 0.003554830
## 7  1e+00   1.0 0.005088595 0.003419971
## 8  1e+01   1.0 0.004171472 0.002575668
## 9  1e+02   1.0 0.004020246 0.002478081
## 10 1e+03   1.0 0.004020246 0.002478081
## 11 1e-01   2.0 0.005221021 0.003561697
## 12 1e+00   2.0 0.005156412 0.003485136
## 13 1e+01   2.0 0.004615067 0.002875331
## 14 1e+02   2.0 0.004464833 0.002720820
## 15 1e+03   2.0 0.004464833 0.002720820
## 16 1e-01   3.0 0.005224336 0.003565170
## 17 1e+00   3.0 0.005188968 0.003518940
## 18 1e+01   3.0 0.004880770 0.003109513
## 19 1e+02   3.0 0.004771591 0.002966224
## 20 1e+03   3.0 0.004771591 0.002966224
## 21 1e-01   4.0 0.005226079 0.003567131
## 22 1e+00   4.0 0.005206265 0.003538252
## 23 1e+01   4.0 0.005033969 0.003267118
## 24 1e+02   4.0 0.004966113 0.003157446
## 25 1e+03   4.0 0.004966113 0.003157446
```

```
best_model <- tune_svm3$best.model
summary(best_model)
```

```
##
## Call:
## best.tune(method = svm, train.x = Class ~ ., data = vald, ranges = list(cost = c(0.1,
##     1, 10, 100, 1000), gamma = c(0.5, 1, 2, 3, 4)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  100
##       gamma:  0.5
##     epsilon:  0.1
##
##
## Number of Support Vectors:  3133
```

```
pred <- predict(best_model, newdata=test)
acc_svm4 <- mean(pred==test$Class)
```