Support-Vector Machines

6.1 Look at the help page for the dataset to find out what the different columns mean

```
library('e1071')
library('kernlab')
```

```
##
## Attaching package: 'kernlab'
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
data(spam)
dim(spam)
```

```
## [1] 4601   58
```

```
head(spam)
```

```
##   make address  all num3d  our over remove internet order mail receive wil
l
## 1 0.00    0.64 0.64     0 0.32 0.00   0.00     0.00  0.00 0.00    0.00 0.6
4
## 2 0.21    0.28 0.50     0 0.14 0.28   0.21     0.07  0.00 0.94    0.21 0.7
9
## 3 0.06    0.00 0.71     0 1.23 0.19   0.19     0.12  0.64 0.25    0.38 0.4
5
## 4 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31 0.3
1
## 5 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31 0.3
1
## 6 0.00    0.00 0.00     0 1.85 0.00   0.00     1.85  0.00 0.00    0.00 0.0
0
##   people report addresses free business email  you credit your font num000
## 1   0.00   0.00      0.00 0.32     0.00  1.29 1.93   0.00 0.96    0   0.00
## 2   0.65   0.21      0.14 0.14     0.07  0.28 3.47   0.00 1.59    0   0.43
## 3   0.12   0.00      1.75 0.06     0.06  1.03 1.36   0.32 0.51    0   1.16
## 4   0.31   0.00      0.00 0.31     0.00  0.00 3.18   0.00 0.31    0   0.00
## 5   0.31   0.00      0.00 0.31     0.00  0.00 3.18   0.00 0.31    0   0.00
## 6   0.00   0.00      0.00 0.00     0.00  0.00 0.00   0.00 0.00    0   0.00
```

```
##   money hp hpl george num650 lab labs telnet num857 data num415 num85
## 1  0.00  0   0      0      0   0    0      0      0    0      0     0
## 2  0.43  0   0      0      0   0    0      0      0    0      0     0
## 3  0.06  0   0      0      0   0    0      0      0    0      0     0
## 4  0.00  0   0      0      0   0    0      0      0    0      0     0
## 5  0.00  0   0      0      0   0    0      0      0    0      0     0
## 6  0.00  0   0      0      0   0    0      0      0    0      0     0
##   technology num1999 parts pm direct cs meeting original project   re  edu
## 1          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
## 2          0    0.07     0  0   0.00  0       0     0.00       0 0.00 0.00
## 3          0    0.00     0  0   0.06  0       0     0.12       0 0.06 0.06
## 4          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
## 5          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
## 6          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
##   table conference charSemicolon charRoundbracket charSquarebracket
## 1     0          0          0.00            0.000                 0
## 2     0          0          0.00            0.132                 0
## 3     0          0          0.01            0.143                 0
## 4     0          0          0.00            0.137                 0
## 5     0          0          0.00            0.135                 0
## 6     0          0          0.00            0.223                 0
##   charExclamation charDollar charHash capitalAve capitalLong capitalTotal
type
## 1           0.778      0.000    0.000      3.756          61          278
spam
## 2           0.372      0.180    0.048      5.114         101         1028
spam
## 3           0.276      0.184    0.010      9.821         485         2259
spam
## 4           0.137      0.000    0.000      3.537          40          191
spam
## 5           0.135      0.000    0.000      3.537          40          191
spam
## 6           0.000      0.000    0.000      3.000          15           54
spam
```

```
str(spam)
```

```
## 'data.frame':    4601 obs. of  58 variables:
```

```
##  $ make            : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
##  $ address         : num  0.64 0.28 0 0 0 0 0 0 0 0.12 ...
##  $ all             : num  0.64 0.5 0.71 0 0 0 0 0 0.46 0.77 ...
##  $ num3d           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ our             : num  0.32 0.14 1.23 0.63 0.63 1.85 1.92 1.88 0.61 0.
19 ...
##  $ over            : num  0 0.28 0.19 0 0 0 0 0 0 0.32 ...
##  $ remove          : num  0 0.21 0.19 0.31 0.31 0 0 0 0.3 0.38 ...
##  $ internet        : num  0 0.07 0.12 0.63 0.63 1.85 0 1.88 0 0 ...
##  $ order           : num  0 0 0.64 0.31 0.31 0 0 0 0.92 0.06 ...
##  $ mail            : num  0 0.94 0.25 0.63 0.63 0 0.64 0 0.76 0 ...
##  $ receive         : num  0 0.21 0.38 0.31 0.31 0 0.96 0 0.76 0 ...
##  $ will            : num  0.64 0.79 0.45 0.31 0.31 0 1.28 0 0.92 0.64 ...
##  $ people          : num  0 0.65 0.12 0.31 0.31 0 0 0 0 0.25 ...
##  $ report          : num  0 0.21 0 0 0 0 0 0 0 0 ...
##  $ addresses       : num  0 0.14 1.75 0 0 0 0 0 0 0.12 ...
##  $ free            : num  0.32 0.14 0.06 0.31 0.31 0 0.96 0 0 0 ...
##  $ business        : num  0 0.07 0.06 0 0 0 0 0 0 0 ...
##  $ email           : num  1.29 0.28 1.03 0 0 0 0.32 0 0.15 0.12 ...
##  $ you             : num  1.93 3.47 1.36 3.18 3.18 0 3.85 0 1.23 1.67 ...
##  $ credit          : num  0 0 0.32 0 0 0 0 0 3.53 0.06 ...
##  $ your            : num  0.96 1.59 0.51 0.31 0.31 0 0.64 0 2 0.71 ...
##  $ font            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num000          : num  0 0.43 1.16 0 0 0 0 0 0 0.19 ...
##  $ money           : num  0 0.43 0.06 0 0 0 0 0 0.15 0 ...
##  $ hp              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ hpl             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ george          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num650          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ lab             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ labs            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ telnet          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num857          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ data            : num  0 0 0 0 0 0 0 0 0.15 0 ...
##  $ num415          : num  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ num85            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ technology       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num1999          : num  0 0.07 0 0 0 0 0 0 0 0 ...
##  $ parts            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ pm               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ direct           : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ cs               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ meeting          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ original         : num  0 0 0.12 0 0 0 0 0 0.3 0 ...
##  $ project          : num  0 0 0 0 0 0 0 0 0 0.06 ...
##  $ re               : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ edu              : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ table            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ conference       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ charSemicolon    : num  0 0 0.01 0 0 0 0 0 0 0.04 ...
##  $ charRoundbracket : num  0 0.132 0.143 0.137 0.135 0.223 0.054 0.206 0.2
71 0.03 ...
##  $ charSquarebracket: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ charExclamation  : num  0.778 0.372 0.276 0.137 0.135 0 0.164 0 0.181 0
.244 ...
##  $ charDollar       : num  0 0.18 0.184 0 0 0 0.054 0 0.203 0.081 ...
##  $ charHash         : num  0 0.048 0.01 0 0 0 0 0 0.022 0 ...
##  $ capitalAve       : num  3.76 5.11 9.82 3.54 3.54 ...
##  $ capitalLong      : num  61 101 485 40 40 15 4 11 445 43 ...
##  $ capitalTotal     : num  278 1028 2259 191 191 ...
##  $ type             : Factor w/ 2 levels "nonspam","spam": 2 2 2 2 2 2 2 2
2 2 ...
```

Spam is a data set collected at Hewlett-Packard Labs, that classifies 4601 e-mails as spam or non-spam. In addition to this class label there are 57 variables indicating the frequency of certain words and characters in the e-mail.

The first 58 variables (columns) contain the frequency of the variable name (e.g., business) in the e-mail. If the variable name starts with num (e.g., num650) the it indicates the frequency of the corresponding number (e.g., 650). The variables 49-54 indicate the frequency of the characters ‘;’, ‘(’, ‘[’, ‘!’, ‘$’, and ‘#’. The variables 55-57 contain the average, longest and total run-length of capital letters. Variable 58 indicates the type of the mail and is either “nonspam” or “spam”, i.e. unsolicited commercial e-mail.

6.2 Fit a support vector classifier using svm() on the training data. type is the target and all

other variables can be used as predictors (hint: you can use the . notation which automatically includes all columns of the data.frame as predictors except the target variable).

```
set.seed(02115)
sample <- sample( c(TRUE, FALSE), nrow(spam), replace=TRUE)
spam$type = as.factor(spam$type)
train <- spam[sample,]
test <- spam[!sample,]
str(spam)
```

```
## 'data.frame':    4601 obs. of  58 variables:
##  $ make             : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
##  $ address          : num  0.64 0.28 0 0 0 0 0 0 0 0.12 ...
##  $ all              : num  0.64 0.5 0.71 0 0 0 0 0 0.46 0.77 ...
##  $ num3d            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ our              : num  0.32 0.14 1.23 0.63 0.63 1.85 1.92 1.88 0.61 0.
19 ...
##  $ over             : num  0 0.28 0.19 0 0 0 0 0 0 0.32 ...
##  $ remove           : num  0 0.21 0.19 0.31 0.31 0 0 0 0.3 0.38 ...
##  $ internet         : num  0 0.07 0.12 0.63 0.63 1.85 0 1.88 0 0 ...
##  $ order            : num  0 0 0.64 0.31 0.31 0 0 0 0.92 0.06 ...
##  $ mail             : num  0 0.94 0.25 0.63 0.63 0 0.64 0 0.76 0 ...
##  $ receive          : num  0 0.21 0.38 0.31 0.31 0 0.96 0 0.76 0 ...
##  $ will             : num  0.64 0.79 0.45 0.31 0.31 0 1.28 0 0.92 0.64 ...
##  $ people           : num  0 0.65 0.12 0.31 0.31 0 0 0 0 0.25 ...
##  $ report           : num  0 0.21 0 0 0 0 0 0 0 0 ...
##  $ addresses        : num  0 0.14 1.75 0 0 0 0 0 0 0.12 ...
##  $ free             : num  0.32 0.14 0.06 0.31 0.31 0 0.96 0 0 0 ...
##  $ business         : num  0 0.07 0.06 0 0 0 0 0 0 0 ...
##  $ email            : num  1.29 0.28 1.03 0 0 0 0.32 0 0.15 0.12 ...
##  $ you              : num  1.93 3.47 1.36 3.18 3.18 0 3.85 0 1.23 1.67 ...
##  $ credit           : num  0 0 0.32 0 0 0 0 0 3.53 0.06 ...
##  $ your             : num  0.96 1.59 0.51 0.31 0.31 0 0.64 0 2 0.71 ...
##  $ font             : num  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ num000           : num  0 0.43 1.16 0 0 0 0 0 0 0.19 ...
##  $ money            : num  0 0.43 0.06 0 0 0 0 0 0.15 0 ...
##  $ hp               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ hpl              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ george           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num650           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ lab              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ labs             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ telnet           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num857           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ data             : num  0 0 0 0 0 0 0 0 0.15 0 ...
##  $ num415           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num85            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ technology       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num1999          : num  0 0.07 0 0 0 0 0 0 0 0 ...
##  $ parts            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ pm               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ direct           : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ cs               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ meeting          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ original         : num  0 0 0.12 0 0 0 0 0 0.3 0 ...
##  $ project          : num  0 0 0 0 0 0 0 0 0 0.06 ...
##  $ re               : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ edu              : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ table            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ conference       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ charSemicolon    : num  0 0 0.01 0 0 0 0 0 0 0.04 ...
##  $ charRoundbracket : num  0 0.132 0.143 0.137 0.135 0.223 0.054 0.206 0.2
71 0.03 ...
##  $ charSquarebracket: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ charExclamation  : num  0.778 0.372 0.276 0.137 0.135 0 0.164 0 0.181 0
.244 ...
##  $ charDollar       : num  0 0.18 0.184 0 0 0 0.054 0 0.203 0.081 ...
##  $ charHash         : num  0 0.048 0.01 0 0 0 0 0 0.022 0 ...
##  $ capitalAve       : num  3.76 5.11 9.82 3.54 3.54 ...
```

```
##  $ capitalLong      : num  61 101 485 40 40 15 4 11 445 43 ...
##  $ capitalTotal     : num  278 1028 2259 191 191 ...
##  $ type             : Factor w/ 2 levels "nonspam","spam": 2 2 2 2 2 2 2 2 2 2 ...
```

```
# Check class distribution in train and test data
table(train$type)
```

```
##
## nonspam    spam
##    1390     880
```

```
table(test$type)
```

```
##
## nonspam    spam
##    1398     933
```

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──
## ✓ tibble  3.1.6     ✓ purrr   0.3.4
## ✓ tidyr   1.1.4     ✓ stringr 1.4.0
## ✓ readr   2.1.0     ✓ forcats 0.5.1
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x kernlab::alpha() masks ggplot2::alpha()
## x purrr::cross()   masks kernlab::cross()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x car::recode()    masks dplyr::recode()
## x purrr::some()    masks car::some()
```

```
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
## The following object is masked from 'package:purrr':
##
##     lift
## The following objects are masked from 'package:Metrics':
```

```
##
##     precision, recall
```

```
# Fit a support vector classifier using svm() on the training data
fit_spam <- svm(type~., train)
summary(fit_spam)
```

```
##
## Call:
## svm(formula = type ~ ., data = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  726
##
##  ( 338 388 )
##
##
## Number of Classes:  2
##
## Levels:
##  nonspam spam
```

```
# use the predict function on the test set predictors
pred_spam = predict(fit_spam,test)
```

```
# 6.3 Calculate classification error rate & accuracy
accuracy <- mean(test$type == pred_spam)
accuracy
```

```
## [1] 0.9232089
```

```
error_rate = mean(test$type != pred_spam)
error_rate
```

```
## [1] 0.07679108
```

```
# Confusion matrix
table(test$type, pred_spam)
```

```
##          pred_spam
##           nonspam spam
##   nonspam    1330   68
##   spam        111  822
```

```
# 6.4 Now fit a support vector classifier again, but select sigmoid for the k
ernel and 100 as the cost parameter. What is the classification error in this
scenario? What does this suggest to you?
fit_spam_new = svm(type~., train, kernel = 'sigmoid', cost = 100)
summary(fit_spam_new)
```

```
##
## Call:
## svm(formula = type ~ ., data = train, kernel = "sigmoid", cost = 100)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  sigmoid
##        cost:  100
##      coef.0:  0
##
## Number of Support Vectors:  379
##
##  ( 191 188 )
##
##
## Number of Classes:  2
##
## Levels:
##  nonspam spam
```

```
pred_spam_new = predict(fit_spam_new,test)
#Calculate classification error rate & accuracy
error_rate_new = mean(test$type != pred_spam_new)
```

```
error_rate_new
## [1] 0.1643072
accuracy_new <- mean(test$type == pred_spam_new)
accuracy_new
## [1] 0.8356928
# Confusion matrix
table(test$type, pred_spam_new)
##          pred_spam_new
##           nonspam spam
##   nonspam    1226  172
##   spam        211  722
# Compare error rates
table(error_rate, error_rate_new)
##                     error_rate_new
## error_rate           0.164307164307164
##   0.0767910767910768                 1
```

The new error rate in 6.4 is 0.1643072 compared to error rate of 0.07679108 in 6.3 This suggests that the default support vector machine model with SVM-Kernel : radial and cost = 1 has better performance than the one with SVM-Kernel: sigmoild and cost = 100, which results in higher accuracy and lower error rate in 6.3 model

the kernel used in training and predicting

- Radial basis function (RBF) Kernel: $K(X,Y)=\exp(\|X-Y\|2/2\sigma2)$ which in simple form can be written as $\exp(-\gamma\cdot\|X-Y\|2),\gamma>0$ RBF uses normal curves around the data points, and sums these so that the decision boundary can be defined by a type of topology condition such as curves where the sum is above a value of 0.5.
- Sigmoid Kernel: $K(X,Y)=\tanh(\gamma\cdot XTY+r)$ which is similar to the sigmoid function in logistic regression.

This suggest that the formula of RBF Kernel works better for this dataset.

Cost is the cost of constraints violation (default: 1)—it is the 'C'-constant of the regularization term in the Lagrange formulation The cost parameter decides how much an SVM should be allowed to "bend" with the data. For a low cost, we aim for a smooth decision surface, which allows more space for error as the penalty is low. For a higher cost, the penalty for misclassifying points is very high, so the decision boundary will perfectly separate the data if possible, which results in higher error rate as the penalty is stricter. Cost is also simply referred to as the cost of misclassification.

Therefore, we can conclude that SVM effectiveness depends upon how you choose the basic 3 requirements: Selection of Kernel, Kernel Parameters, Soft Margin Parameter C (cost) in such a way that it maximises your efficiency, reduces error and overfitting. We have to test different models, changing these

criterias to test which one is more effective as this will be specific for different dataset and purpose of the model.

6.5 How easy is it to interpret the classification performed using svm? Compare the interpretability

of the svm model to that of a regression model (e.g., like the one from the question above)

```
summary(fit_spam)
```

```
##
## Call:
## svm(formula = type ~ ., data = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  726
##
##  ( 338 388 )
##
##
## Number of Classes:  2
##
## Levels:
##  nonspam spam
```

The classification perfomred using SVM is easy to interpret using confusion matrix, classification error, accuracy, sensitivity and specficity metrics.

In regression model, it is necessary to understand the coefficients, R squared metrics and the error metrics like Mean Error, Mean Squared Error (MSE), RMSE Root Mean Square Error, MAPE Mean Absolute Percentage Error, F-statistics and p-value. In addition, we have to also check if there are any linearity anomalies in regression model to make sure that predictors are truly significant as the p-values show.

On the one hand, for SM model, the confusion matrix gives a clear picture about the number observations that are correctly or wrongly classified. Hence it is less demanding and easier to interpret SVM model. Plus, SVM also provides higher accuracy for classification.

On the other hand, while it's true that SVM may come with higher accuracy, Logistics Regression (LR) is much more than just a "classifier" (if we may call it such at all since it predicts a proportion rather than a class). In short, due to the complexity of its statistical interpretation, LR is a parametric/probabilistic method, which produces an inferential and highly interpretable statistical model and, on top of interpretability, it may be used in prediction under certain conditions.

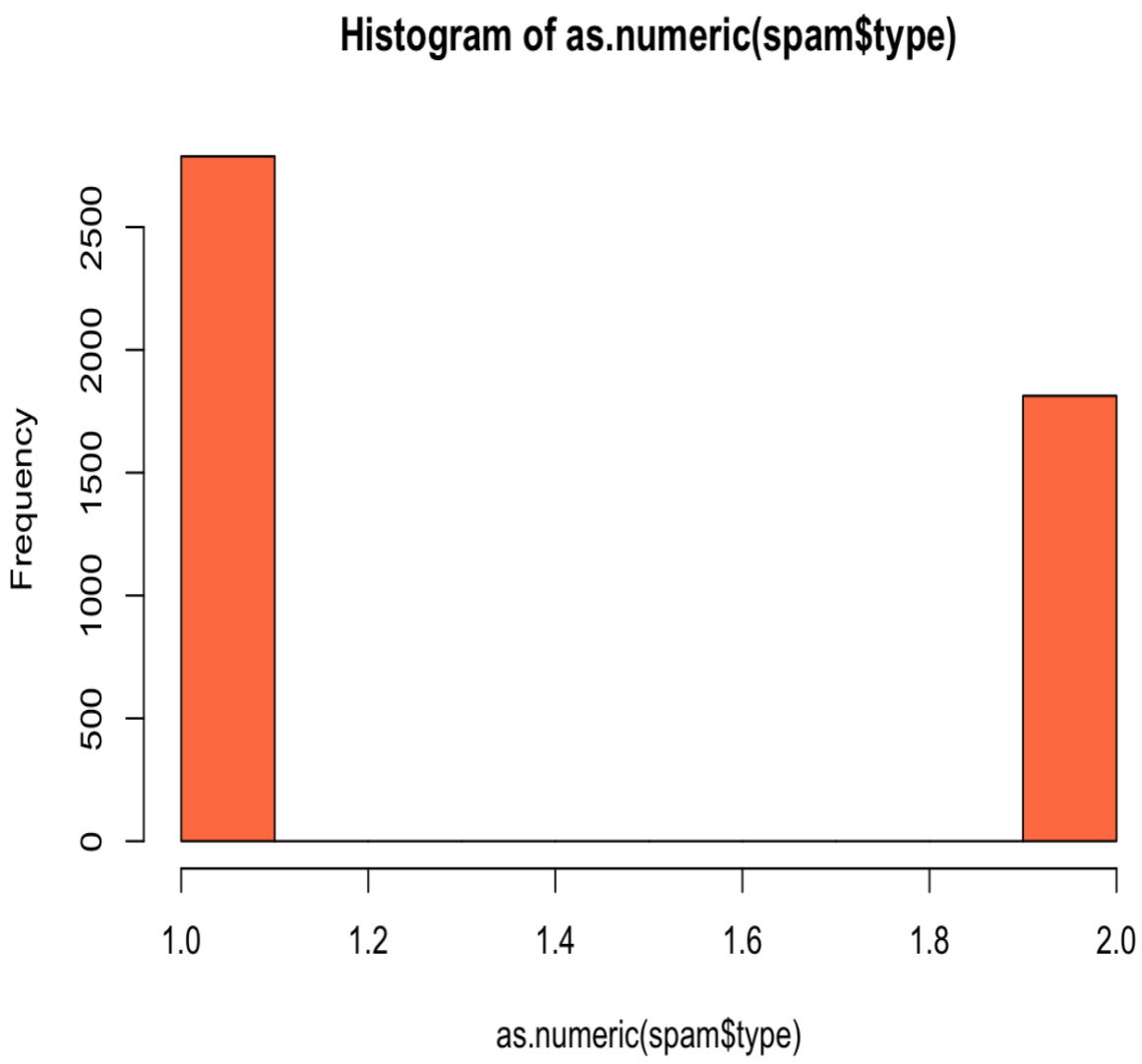
On the other hand, SVM is nonparametric and non-interpretable, and it would be useless in a scenario where we care to explain the behaviour and interactions of variables rather than just finding patterns for prediction.

That said, while there are many alternatives to the predictive accuracy of SVM, I can't think of many to the inferential power of LR.

6.6 (Optional for bonus points) Perform 10 fold cross validation, either writing your own function or using the tune() function to find the best hyper parameter

```
# Check class balance

hist(as.numeric(spam$type),col="coral")
```




```
prop.table(table(spam$type))
```

```
##
##   nonspam      spam
## 0.6059552 0.3940448
```

```
table(spam$type)/nrow(spam)
```

```
##
##   nonspam      spam
## 0.6059552 0.3940448
```

This plot shows that our dataset slightly imbalanced but still good enough. It has a 60:40 ratio so it is good enough. If the dataset has more than 60% of the data in one class. In that case, we can use SMOTE to handle an imbalanced dataset.

```
# The k-Fold
set.seed(100)
# Perform 10 fold cross validation
trctrl <- trainControl(method = "cv", number = 10, savePredictions=TRUE)
nb_fit <- train(factor(type) ~., data = spam, method = "naive_bayes", trContr
ol=trctrl, tuneLength = 0)
nb_fit
```

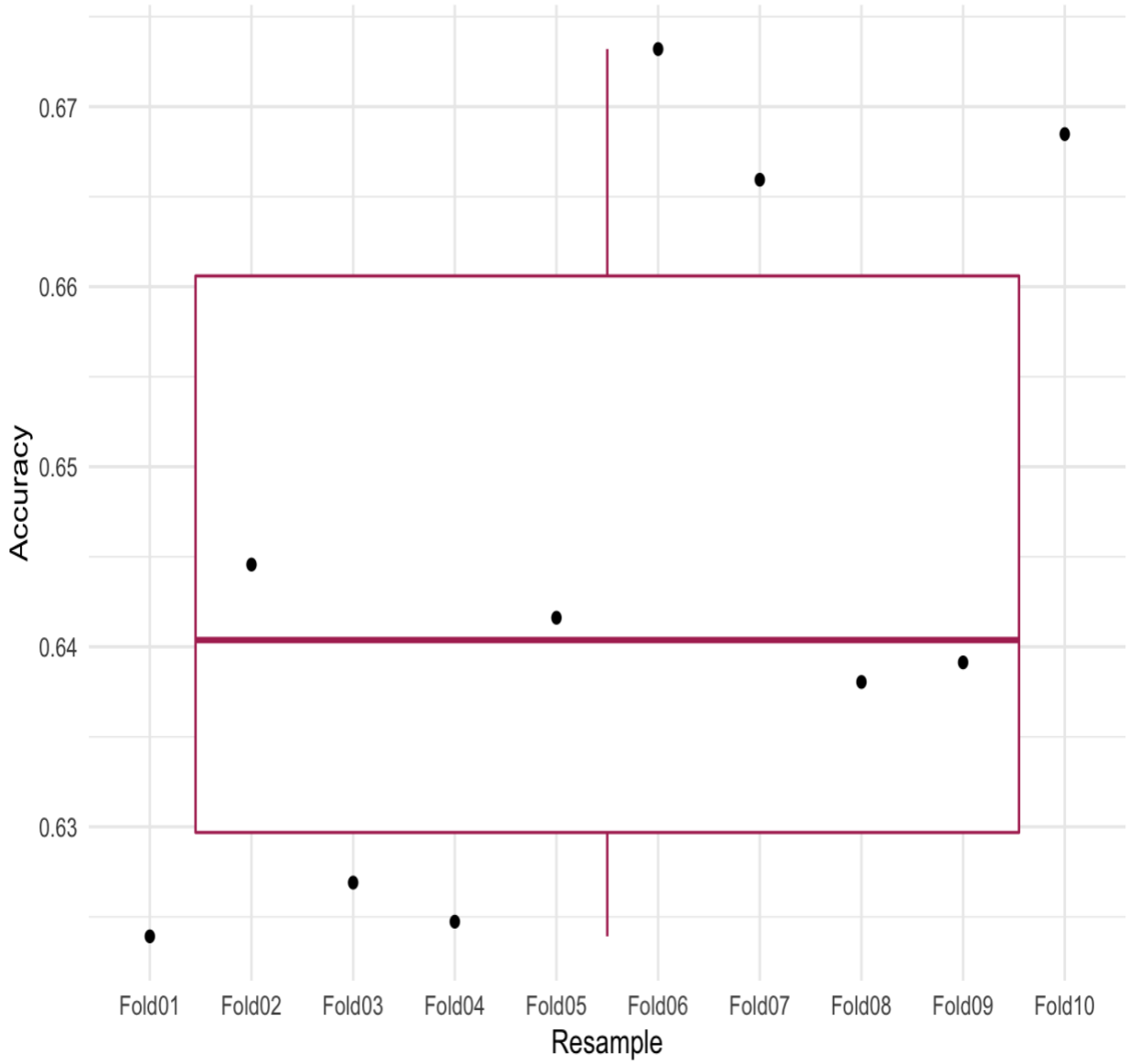
```
## Naive Bayes
##
## 4601 samples
##   57 predictor
##    2 classes: 'nonspam', 'spam'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4141, 4141, 4140, 4140, 4142, 4142, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.7111576  0.4561002
##    TRUE      0.5781455  0.2545793
##
## Tuning parameter 'laplace' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were laplace = 0, usekernel = FALSE
##  and adjust = 1.
```

```
# We can determine that our model is performing well on each fold by looking
at each fold's accuracy
pred <- nb_fit$pred
pred$equal <- ifelse(pred$pred == pred$obs, 1,0)
```

```r
eachfold <- pred %>%
  group_by(Resample) %>%
  summarise_at(vars(equal),
               list(Accuracy = mean))
eachfold
```

```
## # A tibble: 10 × 2
##    Resample Accuracy
##    <chr>       <dbl>
##  1 Fold01      0.624
##  2 Fold02      0.645
##  3 Fold03      0.627
##  4 Fold04      0.625
##  5 Fold05      0.642
##  6 Fold06      0.673
##  7 Fold07      0.666
##  8 Fold08      0.638
##  9 Fold09      0.639
## 10 Fold10      0.668
```

```r
# use the boxplot to represent our accuracies
ggplot(data=eachfold, aes(x=Resample, y=Accuracy, group=1)) +
geom_boxplot(color="maroon") +
geom_point() +
theme_minimal()
```



In the k-fold validation method using Bayes Naives, fold 6 has the highest accuracy (the best hyper parameter) which is 67.32% We can see that each of the folds achieves an accuracy that is not much different from one another. The lowest accuracy is 62.39%, and also in the boxplot, we do not see any outliers. Meaning that our model was performing well across the k-fold cross-validation.

Try hyperameters in Support Vector Machines (SVM)

```
###  Another method
#Set up cross-validation:
library(caret)
library(tictoc)
library(lattice)
```

```r
# Hyperparameters in Support Vector Machines (SVM)
fitControl <- trainControl(method = "repeatedcv", number = 10)
tic()
set.seed(42)
svm_model <- train(type ~ ., data = train ,method = "svmPoly", trControl = fi
tControl, verbose= FALSE)
toc()
```

```
## 81.019 sec elapsed
```

```r
svm_model
```

```
## Support Vector Machines with Polynomial Kernel
##
## 2270 samples
##   57 predictor
##    2 classes: 'nonspam', 'spam'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 2043, 2043, 2043, 2043, 2043, 2043, ...
## Resampling results across tuning parameters:
##
##   degree  scale  C     Accuracy   Kappa
##   1       0.001  0.25  0.7845815  0.5003142
##   1       0.001  0.50  0.8405286  0.6431745
##   1       0.001  1.00  0.8678414  0.7093479
##   1       0.010  0.25  0.8938326  0.7707576
##   1       0.010  0.50  0.9030837  0.7920039
##   1       0.010  1.00  0.9158590  0.8207272
##   1       0.100  0.25  0.9198238  0.8295637
##   1       0.100  0.50  0.9207048  0.8317819
##   1       0.100  1.00  0.9237885  0.8385854
##   2       0.001  0.25  0.8409692  0.6442604
##   2       0.001  0.50  0.8678414  0.7093476
##   2       0.001  1.00  0.8898678  0.7614244
##   2       0.010  0.25  0.9083700  0.8034934
##   2       0.010  0.50  0.9264317  0.8431487
```

```
##   2       0.010  1.00  0.9312775  0.8539957
##   2       0.100  0.25  0.9281938  0.8474384
##   2       0.100  0.50  0.9330396  0.8577746
##   2       0.100  1.00  0.9321586  0.8560874
##   3       0.001  0.25  0.8594714  0.6892632
##   3       0.001  0.50  0.8845815  0.7489619
##   3       0.001  1.00  0.8964758  0.7767628
##   3       0.010  0.25  0.9264317  0.8427416
##   3       0.010  0.50  0.9308370  0.8528541
##   3       0.010  1.00  0.9348018  0.8613843
##   3       0.100  0.25  0.9352423  0.8624181
##   3       0.100  0.50  0.9295154  0.8507667
##   3       0.100  1.00  0.9303965  0.8530268
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were degree = 3, scale = 0.1 and C = 0
.25.
```

```
# Manual hyperparameter tuning in caret
hyperparams <- expand.grid(degree = 4, scale = 1, C = 1)
hyperparams
```

```
##   degree scale C
## 1      4     1 1
```

```
svm_model_2 <- train(type ~ ., data = train, method = "svmPoly", trControl =
fitControl, tuneGrid = hyperparams, verbose = FALSE)
toc()
svm_model_2
```

```
## Support Vector Machines with Polynomial Kernel
##
## 2270 samples
##   57 predictor
##    2 classes: 'nonspam', 'spam'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 2043, 2043, 2043, 2043, 2043, 2043, ...
```

```
## Resampling results:
##
##   Accuracy   Kappa
##   0.8682819  0.7249125
##
## Tuning parameter 'degree' was held constant at a value of 4
## Tuning
##  parameter 'scale' was held constant at a value of 1
## Tuning parameter 'C'
##  was held constant at a value of 1
```

The second method using SVM (svmPoly) formula svm_model contains accross 10 fold resampling has the best hyperparameters of 93.4% compared to 87.58% in Svm_model_2 accuracy.

We can conclude svm_model is the best 10 k-fold cross validation model.