Project Seven for Data Mining

Raymond Anthony Ford
raford2@miners.utep.edu

Due: 14 May 2018

Contents

1 Data preparation

We begin this project by bringing the data into `R`. The data comes from the `kernlab` package, and we seek to use several classification methods to determine if an email is a regular email or a spam email.

```
library(kernlab)
data(spam)
dat <- spam
```

The data set contains 4601 emails in total and of which 1813 were spam emails, so approximately 40% of the emails in the data set are spam. Looking at the data we see that the type of each variable, except for our target variable, is continuous. Our target variable `type` is a character variable that is recorded as either spam or nonspam. Moreover, we note that there are no missing values in our dataset.[1]

We next split the data into a training and test set with the following code.

```
set.seed(5474)
training.data.index <- sample(1:nrow(dat), 0.667*nrow(dat))
```

[1] See the appendix for the ‘R‘ code and output used for this portion.

```
training <- dat[training.data.index, ]
test <- dat[-training.data.index, ]
```

2 Supervised learning

We next fit models using three different statistical learning methods to classify an email as either spam or nonspam; these methods are listed in each of the subsections below.

2.1 Linear discriminant analysis (LDA)

Our first method is linear discriminate analysis (LDA). The code below is used to fit said model.

```
library(MASS)
spam.lda <- lda(type ~ ., data=training, CV=FALSE)
spam.lda.pred <- as.vector(predict(spam.lda, test)$x)
```

2.2 Logistic regression with backward elimination

Our second method is a logistic regression model selected via backward elimination based on BIC. The code below is used to fit this model.

```
fit.full <- suppressWarnings(glm(type~., data=training, family = binomial))
spam.back <- suppressWarnings(step(fit.full, direction = "backward",
                                   k=log(nrow(dat)), trace=FALSE))
spam.back.pred <- predict(spam.back, newdata=test, type="response")
```

2.3 Random Forests

Our final classification model is a random forest. We have elected to use the default values as arguments/parameters when fitting this model. The code for this is below.

```
suppressMessages(library(randomForest, quietly=TRUE))
set.seed(5474)
spam.rf <- randomForest(type~., data=training)
spam.rf.pred <- as.data.frame(predict(spam.rf, test, type="prob"))$spam
```

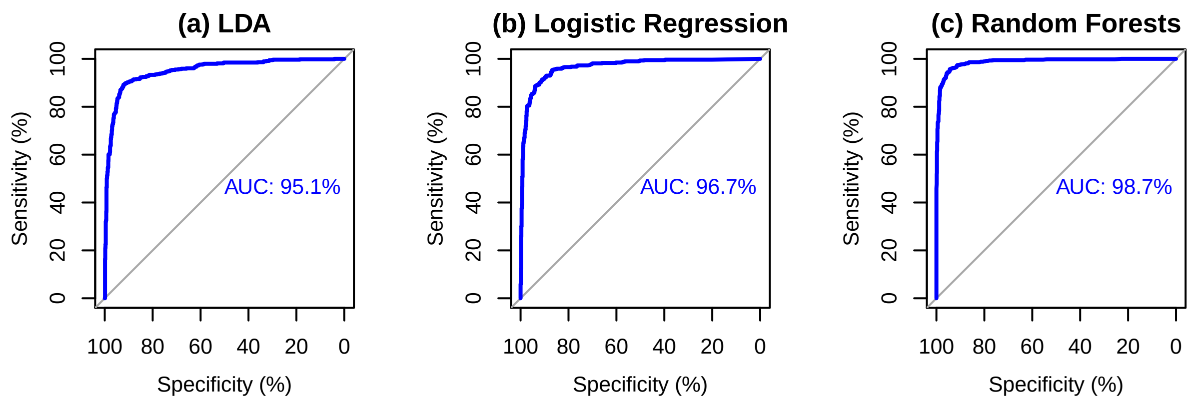

Figure 1: ROC(AUC) curves for each of the classifiction methods employed: linear discriminant analysis (LDA), logistic regression, and random forests, respectively.

2.4 Model comparison

Finally, we evaluate the performance of these three models by comparing their area under the receiver operating characteristic curve (AUC) values.

```
suppressMessages(suppressWarnings(library(pROC, quietly=TRUE)))
par(mfrow=c(1, 3), mar=rep(4,4), pty="s")
plot.roc(test$type, spam.lda.pred,  main="(a) LDA", percent=TRUE,
    print.auc=TRUE, print.auc.cex=1.0, col="blue")
plot.roc(test$type, spam.back.pred,  main="(b) Logistic Regression", percent=TRUE,
    print.auc=TRUE, print.auc.cex=1.0, col="blue")
plot.roc(test$type, spam.rf.pred,  main="(c) Random Forests", percent=TRUE,
    print.auc=TRUE, print.auc.cex=1.0, col="blue")
```

From Figure 1, we see that the random forests method out performs the other methods with an AUC of 98.7%.

3 Additional features from Random Forests

Finally, we train a random forest model with 2000 trees using the entire data set.

```
set.seed(5474)
spam.rf.full <- randomForest(type~., data=dat, importance=TRUE, proximity=TRUE,
                             ntree=2000)
varImpPlot(spam.rf.full, main="Variable importance")
```

From the variable importance plots in Figure 2, we see that `charExclamation` and `charDollar` (the symbols "!" and "$", respectively) are the top two most important variables according to the mean decrease in Gini index. We next plot the partial dependence plots for these two variables.

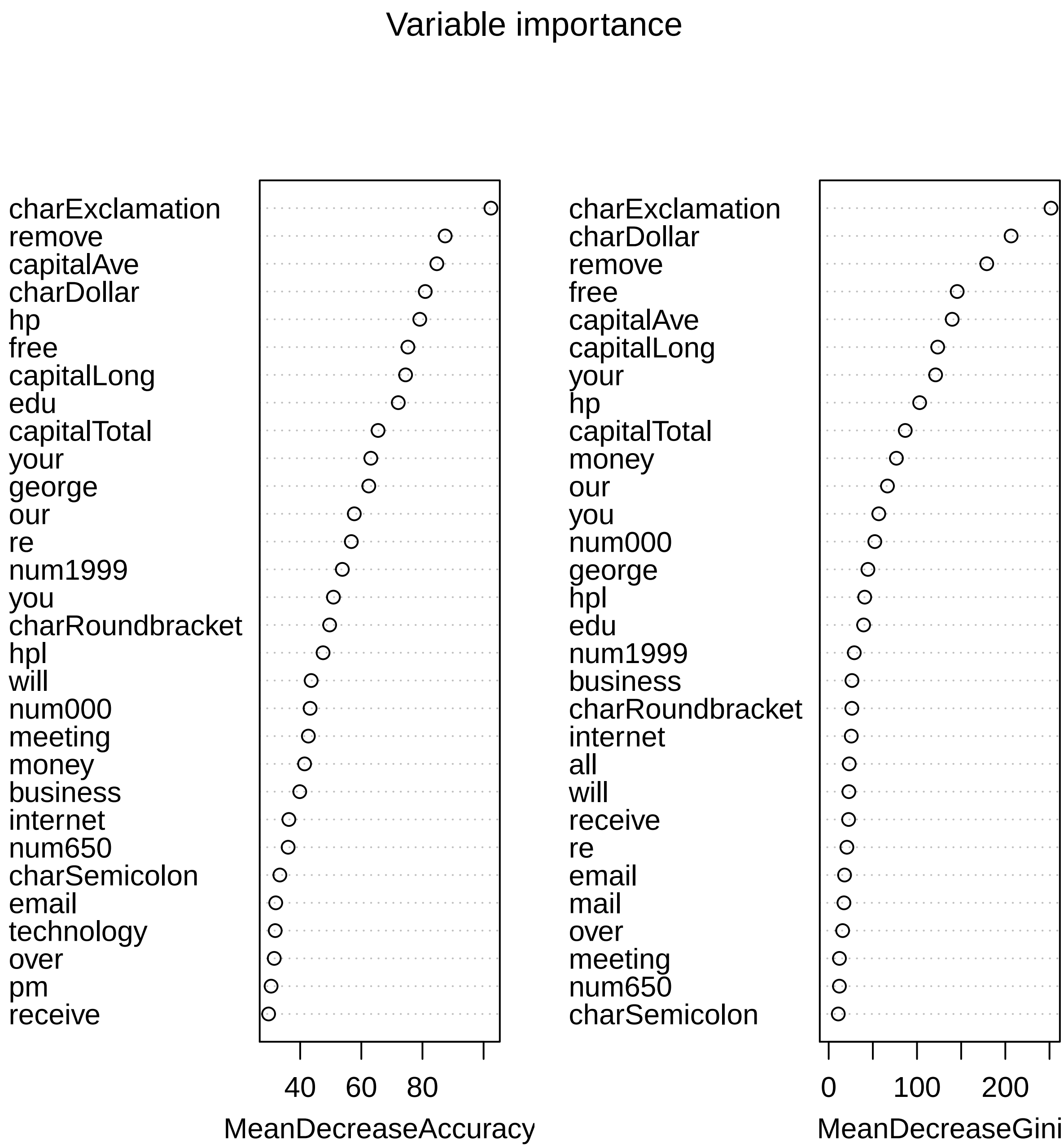

Figure 2: Variable importance plot for the random forest fit with the full data set.

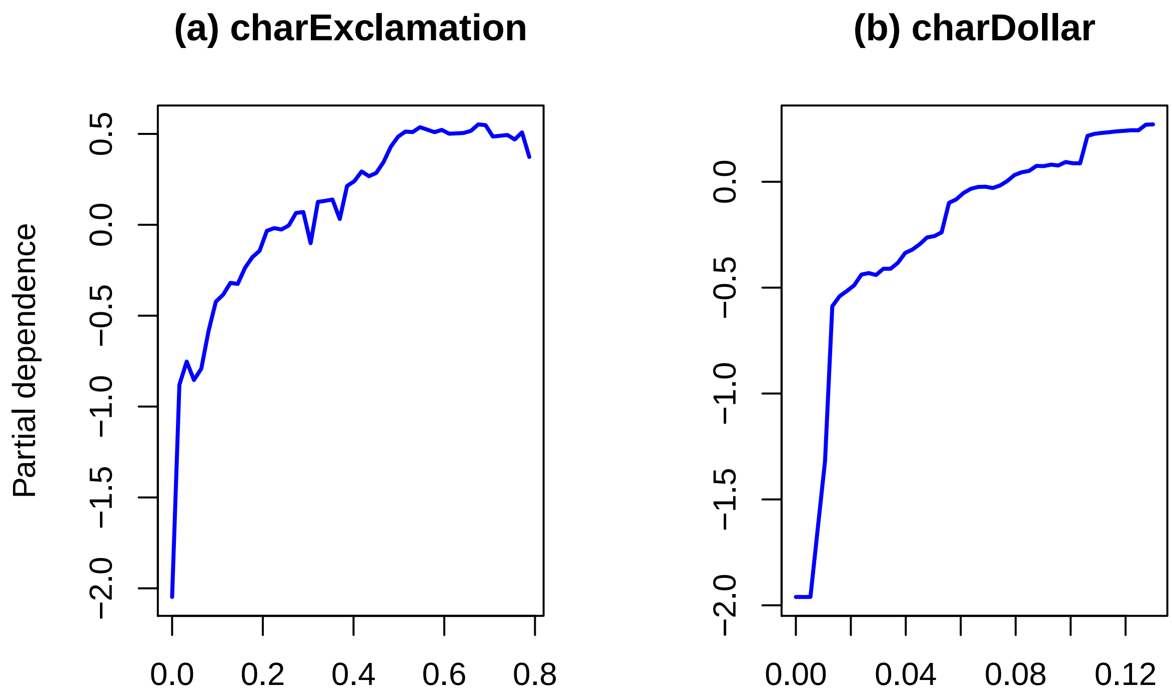

Figure 3: Partial dependence plots for top two important variables.

```
suppressMessages(library(interpretR, quietly=TRUE))
par(mfrow=c(1, 2))
parDepPlot(x.name="charExclamation", spam.rf.full, data=dat,
           main="(a) charExclamation", col="blue", lwd=2, xlab="",
           ylab="Partial dependence")
parDepPlot(x.name="charDollar", spam.rf.full, data=dat,
            main="(b) charDollar", col="blue", lwd=2, xlab="", ylab="")
```

```
library(beepr);beep(8)
```

In the partial dependence plots in Figure 3, we see that as the number of exclamation marks (a) and dollar signs (b) increase,[2] the more likely an email will be classified as a spam email.

Lastly, we transform the proximity matrix into a dissimilarity matrix and plot the first two principal components.

```
DIST <- 1 - spam.rf.full$proximity
labs <- dat[,58]
set.seed(5474)
pca.res <- prcomp(dat[,-58], retx=TRUE)
plot(pca.res$x[,1:2], pch="", main="PC1 and PC2 for email spam data set")
text(pca.res$x[,1:2], col=c("blue", "orange")[labs], lab=labs)
abline(h=0, v=0, lty=2)
```

[2] Outliers are removed in these plots.

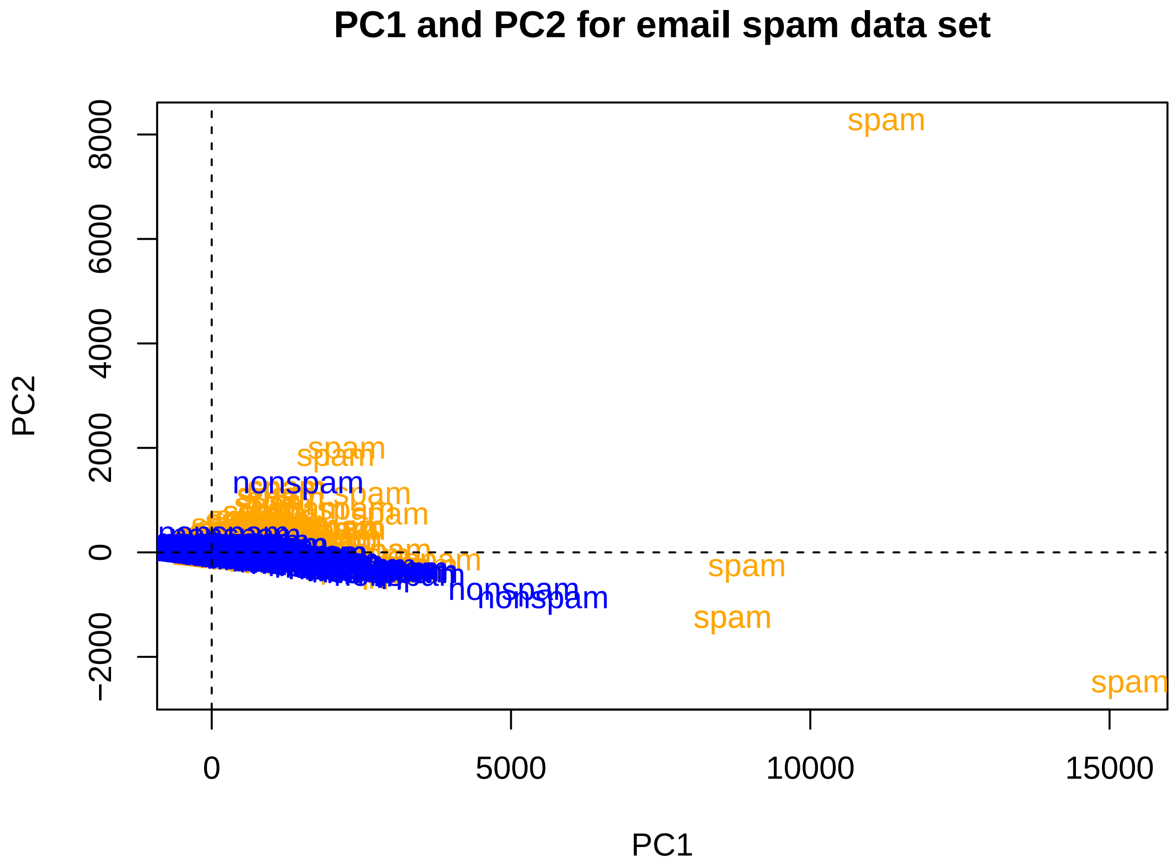

Figure 4: Plot of the spam data set using PCA.

In the above plot, we see that there appears to be a clear clustering of spam and nonspam emails. Moreover, there are many outlying values for the PCs for spam emails, but when these outlying PCs are removed the pattern remains the same.[3]

[3] I have omitted the plot with the outlying PCs removed, but if needed I can email it to you.

A Appendix

Words

```
n <- nrow(dat)
out <- NULL
for (k in 1:ncol(dat)) {
    vname <- colnames(dat)[k]
    x <- as.vector(dat[,k])
    n1 <- sum(is.na(x), na.rm=TRUE)
    n2 <- sum(x=="NA", na.rm=TRUE)
    n3 <- sum(x=='', na.rm=TRUE)
    n4 <- sum(x=="?", na.rm=TRUE)
    n5 <- sum(x=="*", na.rm=TRUE)
    n6 <- sum(x==".", na.rm=TRUE)
    nmiss <- n1 + n2 + n3 + n4 + n5 + n6
    ncomplete <- n - nmiss
    var.type <- typeof(x)
    if (var.type == "integer") {
        if (length(unique(x)) == 2) {
            out <- rbind(out, c(col.number=k, vname=vname, mode="binary",
                                n.levels=length(unique(x)), ncomplete=ncomplete,
                                miss.prop=round(nmiss/n, digits=4)))
        } else {
            out <- rbind(out, c(col.number=k, vname=vname, mode=typeof(x),
                                n.levels=length(unique(x)), ncomplete=ncomplete,
                                miss.prop=round(nmiss/n, digits=4)))
        }
    } else {
        out <- rbind(out, c(col.number=k, vname=vname, mode=typeof(x),
                            n.levels=length(unique(x)), ncomplete=ncomplete,
                            miss.prop=round(nmiss/n, digits=4)))
    }
}
out <- as.data.frame(out)
row.names(out) <- NULL
out
```

```
##   col.number       vname   mode n.levels ncomplete miss.prop
## 1          1        make double      142      4601         0
## 2          2     address double      171      4601         0
## 3          3         all double      214      4601         0
## 4          4       num3d double       43      4601         0
## 5          5         our double      255      4601         0
## 6          6        over double      141      4601         0
```

```
## 7            7           remove   double     173      4601        0
## 8            8         internet   double     170      4601        0
## 9            9            order   double     144      4601        0
## 10          10             mail   double     245      4601        0
## 11          11          receive   double     113      4601        0
## 12          12             will   double     316      4601        0
## 13          13           people   double     158      4601        0
## 14          14           report   double     133      4601        0
## 15          15        addresses   double     118      4601        0
## 16          16             free   double     253      4601        0
## 17          17         business   double     197      4601        0
## 18          18            email   double     229      4601        0
## 19          19              you   double     575      4601        0
## 20          20           credit   double     148      4601        0
## 21          21             your   double     401      4601        0
## 22          22             font   double      99      4601        0
## 23          23           num000   double     164      4601        0
## 24          24            money   double     143      4601        0
## 25          25               hp   double     395      4601        0
## 26          26              hpl   double     281      4601        0
## 27          27           george   double     240      4601        0
## 28          28           num650   double     200      4601        0
## 29          29              lab   double     156      4601        0
## 30          30             labs   double     179      4601        0
## 31          31           telnet   double     128      4601        0
## 32          32           num857   double     106      4601        0
## 33          33             data   double     184      4601        0
## 34          34           num415   double     110      4601        0
## 35          35            num85   double     177      4601        0
## 36          36       technology   double     159      4601        0
## 37          37          num1999   double     188      4601        0
## 38          38            parts   double      53      4601        0
## 39          39               pm   double     163      4601        0
## 40          40           direct   double     125      4601        0
## 41          41               cs   double     108      4601        0
## 42          42          meeting   double     186      4601        0
## 43          43         original   double     136      4601        0
## 44          44          project   double     160      4601        0
## 45          45               re   double     230      4601        0
## 46          46              edu   double     227      4601        0
## 47          47            table   double      38      4601        0
## 48          48       conference   double     106      4601        0
## 49          49    charSemicolon   double     313      4601        0
## 50          50 charRoundbracket   double     641      4601        0
## 51          51 charSquarebracket  double     225      4601        0
```

```
## 52          52   charExclamation    double       964       4601          0
## 53          53        charDollar    double       504       4601          0
## 54          54          charHash    double       316       4601          0
## 55          55        capitalAve    double      2161       4601          0
## 56          56       capitalLong    double       271       4601          0
## 57          57      capitalTotal    double       919       4601          0
## 58          58              type character         2       4601          0
```