Project Two

Ruben Solis
rsolis5@miners.utep.edu

Due: October 11, 2019

Contents

1 Principal Component Analysis (PCA)

1.1 Data Preparation

We begin by reading in the training data into `R`.

```
dat <- read.table(file=
"http://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/optdigits.tra",
                  sep=",", header = FALSE, na.strings = c("NA", "", " "),
                  col.names = c(paste("x", 1:64, sep=""), "digit"))
```

Now, we remove the known value, and then identify the unary variables.

```
dat0 <- data.matrix(dat[order(dat$digit), ]) # Sort for heatmap
labs <- dat0[, c(65)] # Labels for plotting needed later
dat0 <- dat0[,-c(65)] # Remove the known digits
uniq <- apply(dat0, 2, unique)
for (k in 1:length(uniq)) {
    if (length(unique(dat0[,k])) == 1)
        print(paste("x", k))
}
```

```
## [1] "x 1"
```

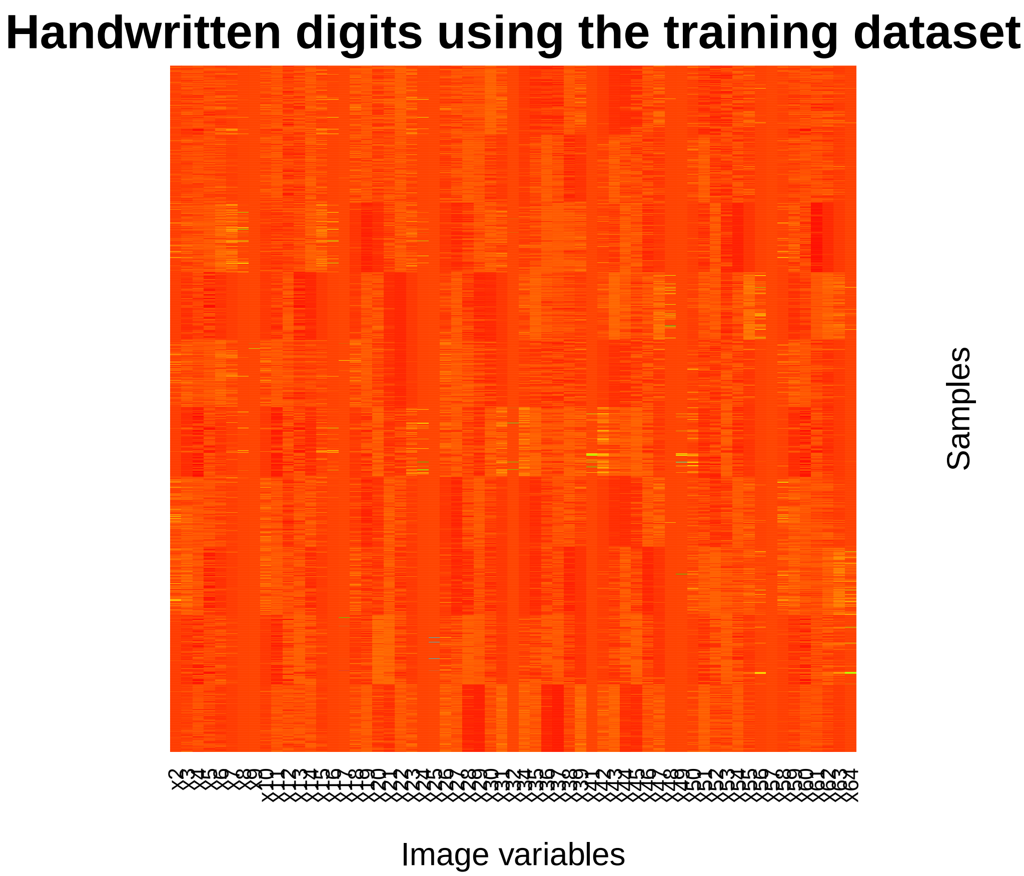


Figure 1: A heatmap of the handwritten digit data.

```
## [1] "x 40"
```

We see from the above output that `x1` and `x40` are unary variables. Futhermore, we also know that `x33` is unary in the testing dataset, so we remove them from the data.

```r
dat0 <- dat0[, -c(1, 33, 40)]
```

Now, we make a heatmap of the data.

```r
n <- NROW(dat0)
color <- rainbow(n, alpha=0.8)
heatmap(dat0, col=color, scale="column", Rowv=NA, Colv=NA,
        labRow=FALSE, margins=c(4,4), xlab="Image variables", ylab="Samples",
        main="Handwritten digits using the training dataset")
```

In Figure 1, we see that there are ten distinct regions. This makes sense as the digits go from zero to ten.

1.2 Classical PCA

We perform classical PCA using the `R` command `prcomp()`.

```
dat0.scaled <- data.frame(apply(dat0, 2, scale, center=T, scale=T))
pca.res <- prcomp(dat0.scaled, retx=TRUE)
```

We next create a screeplot and a plot of the cumulative proportion of variance explained by the number of principal components (PCs) used.

```
par(mfrow=c(2, 1))
screeplot(pca.res, type="line", main="Screeplot for the training data"); box()
sd.pc <- pca.res$sdev
var.pc <- sd.pc**2
prop.pc <- var.pc/sum(var.pc)
plot(cumsum(prop.pc), type="h", xlab="Principal Component (PC)",
    ylab="Cumulative Proportion (CP)", main="Plot of PC vs CP for PCA", xlim=c(0,33))
abline(h=0.7, col='red')
abline(h=0.9, col='blue')
```

In Figure 2, we can see that it would be difficult to select the number of PCs to be used if we only relied on the screeplot. The cumulative proportion plot is more informative, and shows that we should use the first 16 PCs to explain 70% of the variation and 32 PCs if we wanted to explain 90% of the variation.

1.3 Kernal PCA

Next, we will use kernal PCA on the training data.

```
library(kernlab)
kpc <- kpca(~.,data=dat0.scaled, kernel="rbfdot", kpar=list(sigma=0.01),features=20)
```

Now, we next create a screeplot and a plot of the cumulative proportion of variance explained by the number of principal components (PCs) used.

```
par(mfrow=c(2, 1))
var.pc <- eig(kpc)
names(var.pc) <- 1:length(var.pc)
prop.pc <- var.pc/sum(var.pc)
plot(prop.pc, xlab = "Principal Component", main="Screeplot for kPCA",
     ylab = "Proportion of Variance Explained", type = "b")
plot(cumsum(prop.pc), xlab = "Principal Component",
     main=" Plot of PC vs CP for kPCA",
     ylab = "Cumulative Proportion (CP)", type = "h", xlim=c(0, 15))
abline(h=0.7, col='red')
abline(h=0.9, col='blue')
```

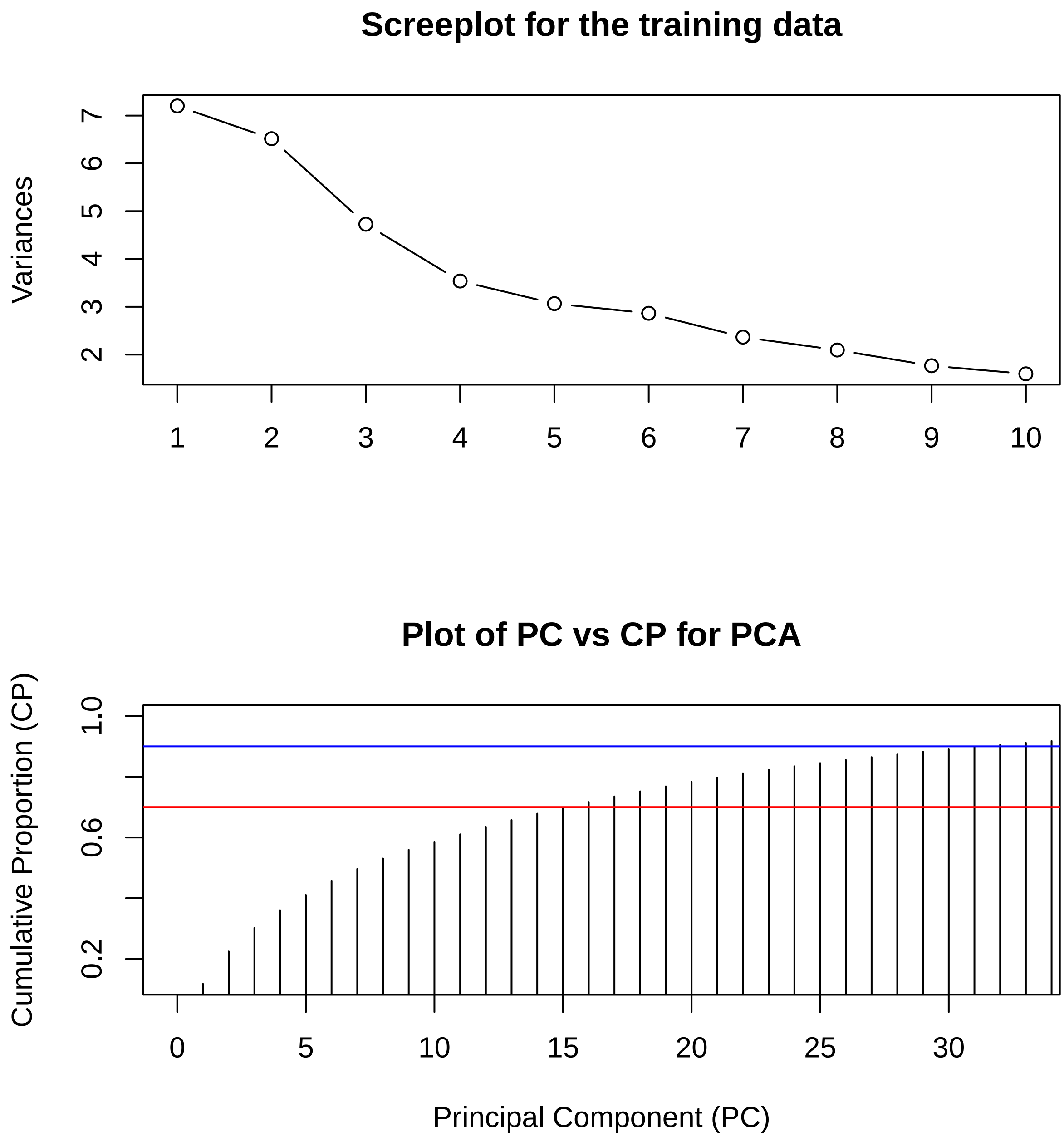
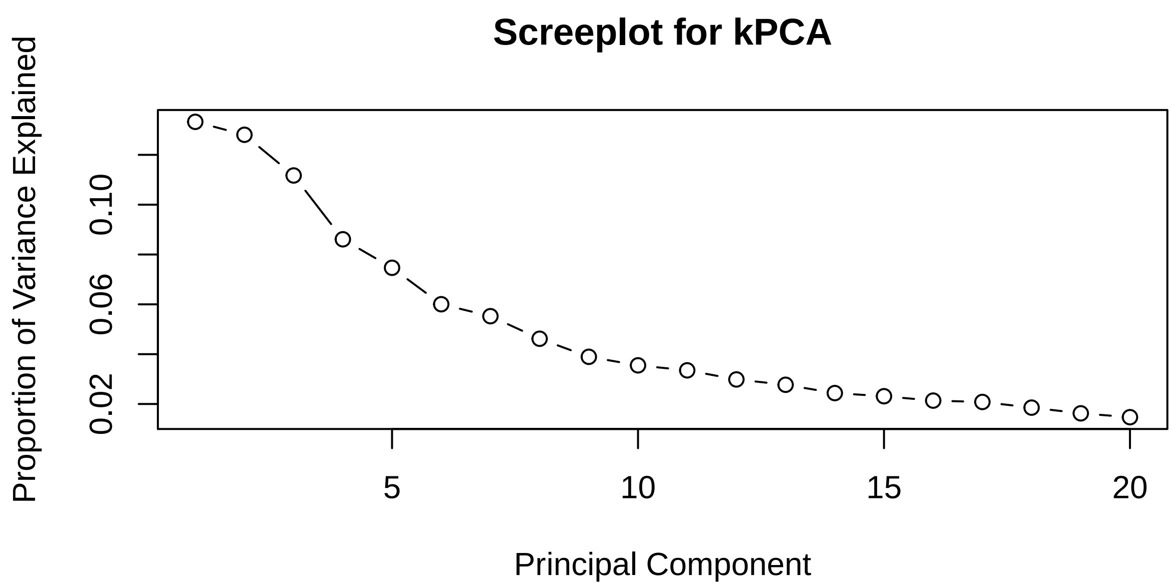


Figure 2: Screeplot and cumulative proportion plot for classical PCA using the handwritten digit training data.



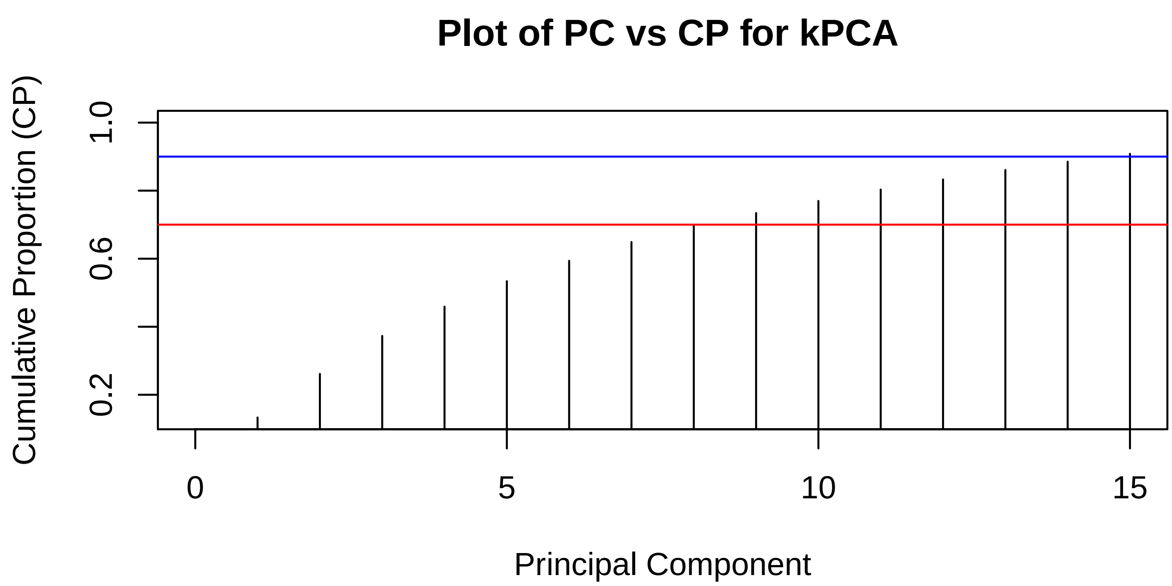


Figure 3: Screeplot and cumulative proportion plot for kernal PCA using the handwritten digit training data.

In Figure 3, we see that it would be difficult to select the number of PCs to be used if we only relied on the screeplot. Again, the cumulative proportion plot is more informative, and shows that we should use the first 9 PCs to explain 70% of the variation and 15 PCs if we wanted to explain 90% of the variation.

1.4 Using a testing dataset

We bring the testing dataset into `R` and prepare the data to use the `predict()` function.

```
dat.test <- read.table(file=
"http://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/optdigits.tes",
                      sep=",", header = FALSE, na.strings = c("NA", "", " "),
                      col.names = c(paste("x", 1:64, sep=""), "digit"))
dat.test <- data.matrix(dat.test[order(dat.test$digit), ])
labs.test <- dat.test[, 65]
dat.test <- dat.test[, -c(1, 33, 40, 65)]
dat.test.scaled <- data.frame(apply(dat.test, 2, scale, center=TRUE, scale=TRUE))
```

Now we compute the predictions with the following code.

```
pred.pca <- predict(pca.res, dat.test.scaled)
pred.kpca <- predict(kpc, dat.test.scaled)
```

Finally, we create a plot that shows the results from the training and testing data using both PCA and kPCA.

```
par(mfrow=c(2, 2))
plot(pca.res$x[,1:2], pch="", main="PCA training data")
text(pca.res$x[,1:2], labels=labs, col=labs+1)
abline(h=0, v=0, lty=2)
plot(rotated(kpc), xlab="PC1",ylab="PC2", main="kPCA training data", pch='')
text(rotated(kpc)[, 1:2], labels=labs, col=labs+1)
abline(h=0, v=0, lty=2)
plot(pred.pca[, 1:2], pch="", main="PCA testing data")
text(pred.pca[, 1:2], labels=labs.test, col=labs.test+1)
abline(h=0, v=0, lty=2)
plot(pred.kpca[, 1:2], pch="", main="kPCA testing data", xlab="PC1", ylab="PC2")
text(pred.kpca[, 1:2], labels=labs.test, col=labs.test+1)
abline(h=0, v=0, lty=2)
```

In Figure 4, we see that the training and testing plots are remarkably similar. Note that the plot for the testing data does not appear as dense as the training data plots. This is because the fact that the number of samples in the testing dataset is less than the number of samples in the training dataset.

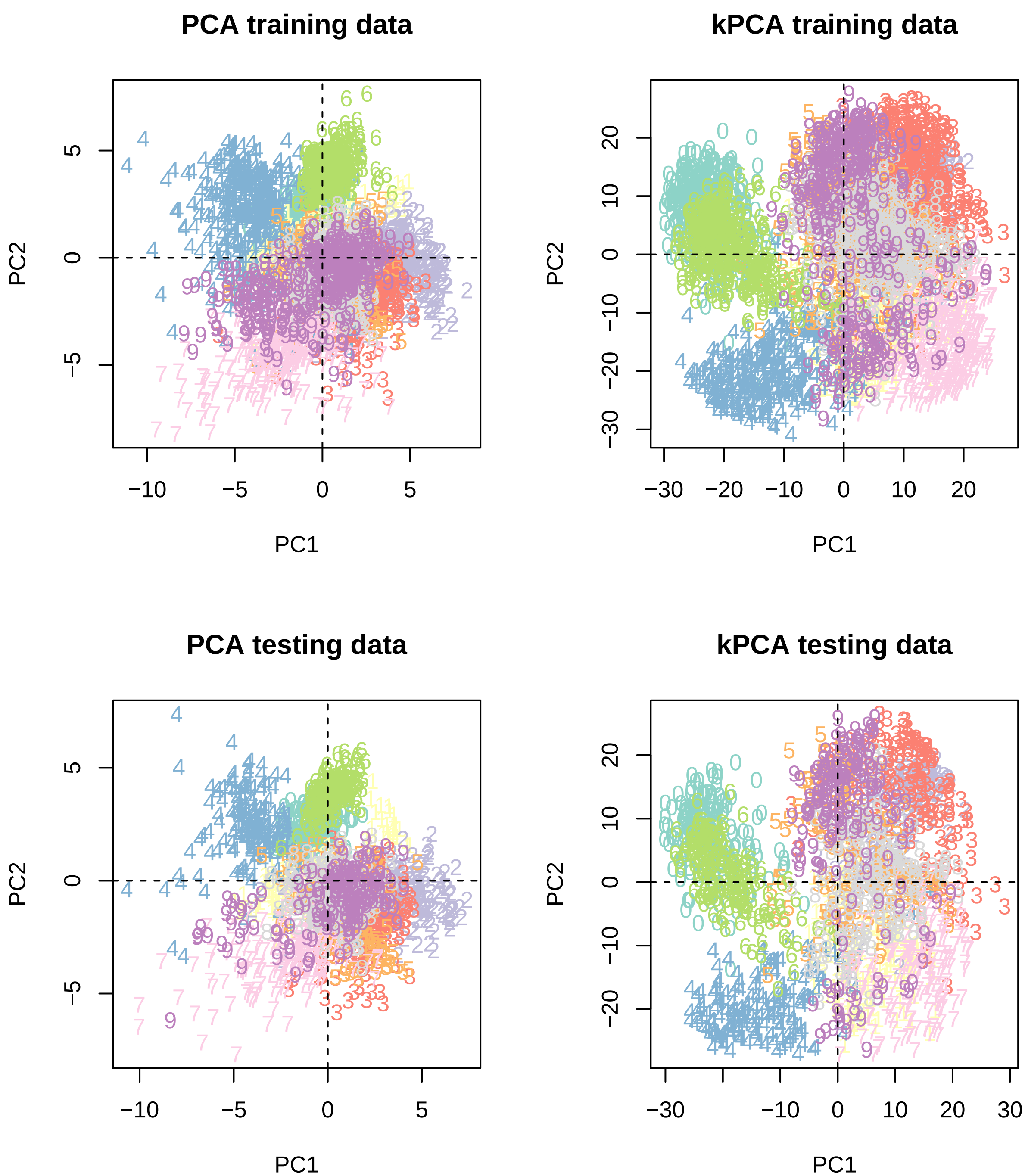


Figure 4: PC1 vs PC2 for the training and testing dataset.

2 Association Rules

We bring the data into R.

```
suppressMessages(library(arules))
```

```
## Warning: package 'arules' was built under R version 3.5.3
```

```
bible <- suppressWarnings(read.transactions(file="AV1611Bible.txt",
                          format="basket", sep =" ", rm.duplicates=FALSE))
dat.bible <- bible
```

Next, we compute some association rules. These parameters where chosen after playing around with different parameterizations.

```
rules <- apriori(dat.bible, parameter=list(support=0.001, confidence=0.2,
    target="rules", minlen=3, maxlen=15), control=list(verbose=FALSE))
rules
```

```
## set of 8080 rules
```

We can visualize the top 10 rules based on both confidence and lift with the following code. The plots are in Figure 5.

```
suppressMessages(library(arulesViz))
```

```
## Warning: package 'arulesViz' was built under R version 3.5.3
```

```
par(mfrow=c(2, 1))
subrules <- head(rules, n=10, by="conf")
subrules2 <- head(rules, n=10, by="lift")
plot(subrules, method="graph", main="Top ten rules based on confidence")
plot(subrules2, method="graph", main="Top ten rules based on lift")
```

Before we output the top five rules for confidence and lift we need to orgainize the association rule output. We do this with the following code.

```
RULES <- as(rules, "data.frame")
rules0 <- data.frame(matrix(unlist(strsplit(as.character(RULES$rules), split="=>")),
                            ncol=2, byrow=TRUE))
colnames(rules0) <- c("LHS", "RHS")
rule.size <- function(x){length(unlist(strsplit(as.character(x), split=",")))}
rules0$size <- apply(rules0, 1, rule.size)
rules0$size[as.character(rules0$LHS)=="{} "] <- rules0$size[as.character(RULES$LHS)=="{
RULES <- cbind(RULES, rules0)
```

The top five rules based on confidence given are below.

```
top5conf <- RULES[order(-RULES$confidence), ][1:5, ]
top5conf
```

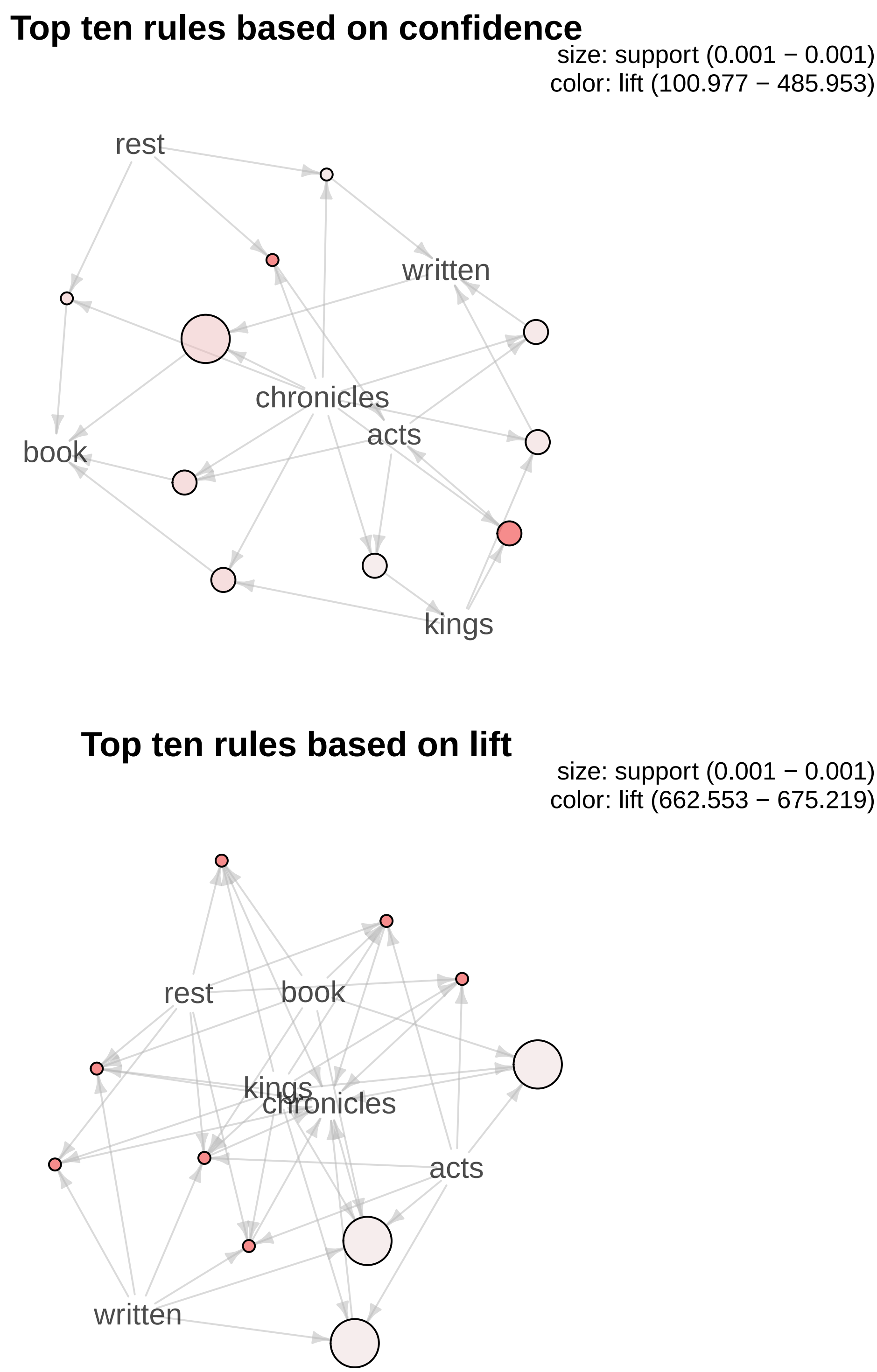


Figure 5: Association rules plot for the bible data.

```
##                             rules     support confidence     lift count
## 1     {acts,chronicles} => {book} 0.001093212          1 179.7746    34
## 4  {acts,chronicles} => {written} 0.001093212          1 118.2548    34
## 8     {chronicles,rest} => {acts} 0.001061059          1 485.9531    33
## 10   {acts,chronicles} => {kings} 0.001093212          1 100.9773    34
## 11   {chronicles,kings} => {acts} 0.001093212          1 485.9531    34
##                     LHS         RHS size
## 1   {acts,chronicles}      {book}    3
## 4   {acts,chronicles}   {written}    3
## 8   {chronicles,rest}      {acts}    3
## 10  {acts,chronicles}     {kings}    3
## 11 {chronicles,kings}      {acts}    3
```

The top five rules based on lift are below.

```
top5lift <- RULES[order(-RULES$lift), ][1:5, ]
top5lift
```

```
##                                               rules     support confidence
## 5509         {acts,kings,rest} => {chronicles} 0.001061059      0.825
## 5521         {book,kings,rest} => {chronicles} 0.001061059      0.825
## 5525      {kings,rest,written} => {chronicles} 0.001061059      0.825
## 7637    {acts,book,kings,rest} => {chronicles} 0.001061059      0.825
## 7642 {acts,kings,rest,written} => {chronicles} 0.001061059      0.825
##          lift count                        LHS          RHS size
## 5509 675.2191    33         {acts,kings,rest}  {chronicles}    4
## 5521 675.2191    33         {book,kings,rest}  {chronicles}    4
## 5525 675.2191    33      {kings,rest,written}  {chronicles}    4
## 7637 675.2191    33    {acts,book,kings,rest}  {chronicles}    5
## 7642 675.2191    33 {acts,kings,rest,written}  {chronicles}    5
```

From the above outputs, it does not appear that the top fives rules for both measures are interesting. In Figure 5 we see additional rules and their relationships, but the results are again not interesting, nor enlightening. It may have better to limit the scope of the project to only one book from the bible rather than use the entire bible, as we may have found more interesting rules.

The biggest problem with both confidence and lift is that they are symmetric. That is we have $conf(A \to B) = conf(B \to A)$ and $lift(A \to B) = lift(B \to A)$, whereas conviction is not and is thus direction sensitive.

3 Appendix

3.1 PCA data summary for training dataset

```
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    x1 integer        1      3823         0
## 2          2    x2 integer        9      3823         0
## 3          3    x3 integer       17      3823         0
## 4          4    x4 integer       17      3823         0
```

```
## 5          5    x5 integer       17      3823          0
## 6          6    x6 integer       17      3823          0
## 7          7    x7 integer       17      3823          0
## 8          8    x8 integer       16      3823          0
## 9          9    x9 integer        4      3823          0
## 10        10   x10 integer       16      3823          0
## 11        11   x11 integer       17      3823          0
## 12        12   x12 integer       17      3823          0
## 13        13   x13 integer       17      3823          0
## 14        14   x14 integer       17      3823          0
## 15        15   x15 integer       17      3823          0
## 16        16   x16 integer       15      3823          0
## 17        17   x17 integer        5      3823          0
## 18        18   x18 integer       17      3823          0
## 19        19   x19 integer       17      3823          0
## 20        20   x20 integer       17      3823          0
## 21        21   x21 integer       17      3823          0
## 22        22   x22 integer       17      3823          0
## 23        23   x23 integer       17      3823          0
## 24        24   x24 integer        9      3823          0
## 25        25   x25  binary        2      3823          0
## 26        26   x26 integer       17      3823          0
## 27        27   x27 integer       17      3823          0
## 28        28   x28 integer       17      3823          0
## 29        29   x29 integer       17      3823          0
## 30        30   x30 integer       17      3823          0
## 31        31   x31 integer       17      3823          0
## 32        32   x32 integer        3      3823          0
## 33        33   x33  binary        2      3823          0
## 34        34   x34 integer       16      3823          0
## 35        35   x35 integer       17      3823          0
## 36        36   x36 integer       17      3823          0
## 37        37   x37 integer       17      3823          0
## 38        38   x38 integer       17      3823          0
## 39        39   x39 integer       15      3823          0
## 40        40   x40 integer        1      3823          0
## 41        41   x41 integer        8      3823          0
## 42        42   x42 integer       17      3823          0
## 43        43   x43 integer       17      3823          0
## 44        44   x44 integer       17      3823          0
## 45        45   x45 integer       17      3823          0
## 46        46   x46 integer       17      3823          0
## 47        47   x47 integer       17      3823          0
## 48        48   x48 integer        5      3823          0
## 49        49   x49 integer        8      3823          0
```

```
## 50          50   x50 integer       17       3823          0
## 51          51   x51 integer       17       3823          0
## 52          52   x52 integer       17       3823          0
## 53          53   x53 integer       17       3823          0
## 54          54   x54 integer       17       3823          0
## 55          55   x55 integer       17       3823          0
## 56          56   x56 integer       11       3823          0
## 57          57   x57  binary        2       3823          0
## 58          58   x58 integer       11       3823          0
## 59          59   x59 integer       17       3823          0
## 60          60   x60 integer       17       3823          0
## 61          61   x61 integer       17       3823          0
## 62          62   x62 integer       17       3823          0
## 63          63   x63 integer       17       3823          0
## 64          64   x64 integer       16       3823          0
## 65          65 digit integer       10       3823          0
```