Thomas Huelsnitz
ISYE 6501
Week 4 Homework

HW04

9.1

Using the same crime data set uscrime.txt as in Question 8.2, apply Principal Component Analysis and then create a regression model using the first few principal components. Specify your new model in terms of the original variables (not the principal components), and compare its quality to that of your solution to Question 8.2. You can use the R function prcomp for PCA. (Note that to first scale the data, you can include scale. = TRUE to scale as part of the PCA function. Don't forget that, to make a prediction for the new city, you'll need to unscale the coefficients (i.e., do the scaling calculation in reverse)!)

Begin by loading the data and performing a PCA on the feature fields.

```
crime <- read.table("9.1uscrimeSummer2018.txt", header=TRUE)

pca <- prcomp(crime[,1:15], scale. = TRUE)
summary(pca)

## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     2.4534 1.6739 1.4160 1.07806 0.97893 0.74377
## Proportion of Variance 0.4013 0.1868 0.1337 0.07748 0.06389 0.03688
## Cumulative Proportion  0.4013 0.5880 0.7217 0.79920 0.86308 0.89996
##                            PC7     PC8     PC9    PC10    PC11    PC12
## Standard deviation     0.56729 0.55444 0.48493 0.44708 0.41915 0.35804
## Proportion of Variance 0.02145 0.02049 0.01568 0.01333 0.01171 0.00855
## Cumulative Proportion  0.92142 0.94191 0.95759 0.97091 0.98263 0.99117
##                           PC13   PC14    PC15
## Standard deviation     0.26333 0.2418 0.06793
## Proportion of Variance 0.00462 0.0039 0.00031
## Cumulative Proportion  0.99579 0.9997 1.00000
```

Concatenate the first n_pca_fields to the target value (Crime field) for each data point in the crime df. Using n_pca_fields=7 here because that is the fewest number of components which are responsible for >90% of the variance in the initial fields.

```
n_pca_fields <- 7
pca_crime <- data.frame(cbind(pca$x[,1:n_pca_fields], Crime=crime[,16]))

model <- lm(Crime~., data=pca_crime)
summary(model)

##
## Call:
## lm(formula = Crime ~ ., data = pca_crime)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -475.41 -141.65   34.73  137.25  412.32
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   905.09      34.21  26.454  < 2e-16 ***
## PC1            65.22      14.10   4.626 4.04e-05 ***
## PC2           -70.08      20.66  -3.392   0.0016 **
## PC3            25.19      24.42   1.032   0.3086
## PC4            69.45      32.08   2.165   0.0366 *
## PC5          -229.04      35.33  -6.483 1.11e-07 ***
## PC6           -60.21      46.50  -1.295   0.2029
## PC7           117.26      60.96   1.923   0.0617 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 234.6 on 39 degrees of freedom
## Multiple R-squared:  0.6882, Adjusted R-squared:  0.6322
## F-statistic:  12.3 on 7 and 39 DF,  p-value: 3.513e-08
```

Converting the PCA coefficients to their equivalent in terms of the original fields in the data set. Also have to undo the scaling that was done in the prcomp function prior to the PCA.

```
model_pca_coef <- model$coefficients[2:length(model$coefficients)]
pca_x_comp <- t(pca$rotation[,1:length(model$coefficients) - 1])
scaled_coef <- model_pca_coef%*%pca_x_comp
intercept <- model$coefficients[1] - sum(scaled_coef*sapply(crime[,1:15],
mean)/sapply(crime[,1:15],sd))
coef <- scaled_coef/sapply(crime[,1:15],sd)
```

Now use the de-scaled coefficients and intercept to see what this model calculates for the test point from question 8.2.

```
M = 14.0
So = 0
Ed = 10.0
Po1 = 12.0
Po2 = 15.5
LF = 0.640
M.F = 94.0
Pop = 150
NW = 1.1
U1 = 0.120
U2 = 3.6
Wealth = 3200
Ineq = 20.1
Prob = 0.04
Time = 39.0
testpoint <- data.frame(M,So,Ed, Po1, Po2, LF, M.F, Pop, NW, U1, U2, Wealth,
```

```
Ineq, Prob, Time)
ans <- coef%*%t(as.matrix(testpoint)) + intercept
ans

##          [,1]
## [1,] 1230.418
```

Using 7 out of 15 PCs explains 92% of the variance we see in the fields of interest. Even so, the model built on these PCs gives a substantially different prediction for the new data point: 1230 for the PCA based model versus 155 for the original model from question 8.2. Let's take a look at how the r-squared values for the PCA based model compare to the results from 8.2.

```
summary(model)$r.squared

## [1] 0.6881819

summary(model)$adj.r.squared

## [1] 0.6322145
```

The PCA based model has r-squared and adjusted r-squared values of 0.69 and 0.63 respectively. Compare this to the original model which had 0.80 and 0.71 for these measures. PCA has allowed us to condense the input data set into fewer components, but at the cost of goodness of fit of the model to the data. This will generally be the case with PCA, but in some instances where the data set is massive and/or some of the fields are very highly correlated, the tradeoff between reduced complexity and reduced goodness of fit will be more favorable and PCA will be a useful tool. For a data set of this size and amount of correlation among fields, PCA is probably not very useful.

10.1

Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using (a) a regression tree model, and (b) a random forest model. In R, you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don't just stop when you have a good model, but interpret it too).

Begin by loading the necessary packages and data. Split the data set into a training and a testing set.

```
library(tree)
library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

set.seed(1)
```

```
calc.RMSE <- function(predicted, actual){
  RMSE <- sqrt(mean((predicted - actual)^2))
}

crime <- read.table("9.1uscrimeSummer2018.txt", header=TRUE)
test_ind <- sample(1:nrow(crime), size=12)
crime_train <- crime[-test_ind,]
crime_test <- crime[test_ind,]
```

Using a regression tree model we get the following performance on the training set and on the test set.

```
cart_model <- tree(Crime~., data=crime_train, x=TRUE, y=TRUE)
cart_rmse_train <- calc.RMSE(predict(cart_model), cart_model$y)
cart_mean_abs_e_train <- mean(abs((predict(cart_model) -
cart_model$y)/cart_model$y))
cart_rmse_test <- calc.RMSE(predict(cart_model, crime_test),
crime_test$Crime)
cart_mean_abs_e_test <- mean(abs((predict(cart_model, crime_test) -
crime_test$Crime)/crime_test$Crime))

## Training RMSE: 188.9

## For training data: On average, the model predicted value is within 16.7
percent of the actual value

## Test RMSE: 381.7

## For testing data: On average, the model predicted value is within 34.6
percent of the actual value
```

The large disparity between the performance on the training data and the testing data indicate that the classifier is over fitting the training data.

Using the same training and test set but this time with a random forest classifier yields the following results:

```
numpred <- 4
rf_model <- randomForest(Crime~., data=crime_train, mtry=numpred,
importance=TRUE)
rf_rmse_train <- calc.RMSE(predict(rf_model), rf_model$y)
rf_mean_abs_e_train <- mean(abs((predict(rf_model) - rf_model$y)/rf_model$y))
rf_rmse_test <- calc.RMSE(predict(rf_model, crime_test), crime_test$Crime)
rf_mean_abs_e_test <- mean(abs((predict(rf_model, crime_test) -
crime_test$Crime)/crime_test$Crime))

## RMSE: 292.4

## For training data: On average, the model predicted value is within 28.3
percent of the actual value

## RMSE: 321.5
```

```
## For testing data:On average, the model predicted value is within 28.8
percent of the actual value
```

Here we see that the random forest classifier has similar performance for the training and test data sets. It seems to generalize better than the single regression tree in the first section and offers better results on the test set than the single regression tree does.

These results are consistent with what we expect from these models. The random forest method fits many (500 in this case) trees to bootstrapped samples of the underlying data. Each of these samples is biased, but by aggregating them into a 'forest' of trees, the biases of the individual trees is cancelled out.

I also learned something new about the regression tree when working on this problem. The lecture led me to believe that this function was segmenting the data into leaves and then fitting a regression model to each of these leaves. This, however, is not exactly what is happening as evidenced by the fact that there are only 5 unique model predicted values for the 35 data points in the training data set:

```
length(predict(cart_model))

## [1] 35

length(unique(predict(cart_model)))

## [1] 5
```

The tree model is actually iterating through the sorted lists of values for each feature and splitting such that the combined sum of the squared deviations from the group average for each newly created group is minimized. The predicted value for a point in a leaf is then just the average value of that leaf.

10.2

Describe a situation or problem from your job, everyday life, current events, etc., for which a logistic regression model would be appropriate. List some (up to 5) predictors that you might use.

I am interested in understanding whether or not a candidate will win or lose an election. This is a situation with a binary outcome (0 or 1). Some variables of interest for this might be:

1. Amount of money spent on the campaign.
2. Amount of commercial airtime.
3. Amount of twitter tweets.
4. Whether or not the candidate is an incumbent.
5. How the candidate's party is trending in recent elections in their district/state/country.

Thomas Huelsnitz
ISYE 6501
Week 4 Homework

10.3

1. Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german / (description at http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29), use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors used and their coefficients), the software output, and the quality of fit. You can use the glm function in R. To get a logistic regression (logit) model on data where the response is either zero or one, use family=binomial(link="logit") in your glm function call.

2. Because the model gives a result between 0 and 1, it requires setting a threshold probability to separate between "good" and "bad" answers. In this data set, they estimate that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad. Determine a good threshold probability based on your model.

Begin with loading the data, transforming the response value so that it is between 0 and 1 instead of 1 and 2 and dividing the data into test and train data. After this, a logistic regression model is fitted to the training data and we show a summary of the model statistics and a 10-fold cross validation estimate of prediction error.

```
findat <- data.frame(read.table("10.3germancreditSummer2018.txt", header =
F))
findat[,21] <- findat[,21] - 1


train_indices <- sample(1:nrow(findat), size = round(nrow(findat) * .8))
train = findat[train_indices,]
test = findat[-train_indices,]

func = V21 ~ factor(V1) + V2 + factor(V3) + factor(V4) + V5 + factor(V6) +
factor(V7) + V8 + factor(V9) + factor(V10) + V11 + factor(V12) + V13 +
factor(V14) + factor(V15) + V16 + factor(V17) + V18 + factor(V19) +
factor(V20)
model = glm(func, family = binomial(link = "logit"), data = train)
summary(model)

##
## Call:
## glm(formula = func, family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3385  -0.6612  -0.3510   0.6757   2.5724
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      8.886e-01  1.265e+00   0.703 0.482324
```

```
## factor(V1)A12   -4.362e-01  2.521e-01  -1.730 0.083621 .
## factor(V1)A13   -9.076e-01  4.309e-01  -2.107 0.035151 *
## factor(V1)A14   -1.712e+00  2.671e-01  -6.407 1.48e-10 ***
## V2               4.199e-02  1.094e-02   3.837 0.000124 ***
## factor(V3)A31   -4.550e-02  6.423e-01  -0.071 0.943532
## factor(V3)A32   -8.500e-01  4.954e-01  -1.716 0.086202 .
## factor(V3)A33   -9.172e-01  5.445e-01  -1.684 0.092095 .
## factor(V3)A34   -1.873e+00  5.088e-01  -3.682 0.000231 ***
## factor(V4)A41   -1.544e+00  4.203e-01  -3.673 0.000240 ***
## factor(V4)A410  -2.003e+00  8.600e-01  -2.330 0.019830 *
## factor(V4)A42   -9.706e-01  3.038e-01  -3.195 0.001400 **
## factor(V4)A43   -9.204e-01  2.892e-01  -3.183 0.001460 **
## factor(V4)A44   -4.840e-01  8.053e-01  -0.601 0.547838
## factor(V4)A45    4.333e-01  7.123e-01   0.608 0.542984
## factor(V4)A46    4.481e-02  4.387e-01   0.102 0.918641
## factor(V4)A48   -6.631e-01  1.319e+00  -0.503 0.615142
## factor(V4)A49   -8.625e-01  3.879e-01  -2.224 0.026180 *
## V5               7.627e-05  5.008e-05   1.523 0.127804
## factor(V6)A62   -2.976e-01  3.250e-01  -0.916 0.359868
## factor(V6)A63   -2.605e-01  4.230e-01  -0.616 0.538051
## factor(V6)A64   -1.372e+00  5.572e-01  -2.462 0.013816 *
## factor(V6)A65   -1.422e+00  3.234e-01  -4.398 1.09e-05 ***
## factor(V7)A72    4.394e-02  4.810e-01   0.091 0.927204
## factor(V7)A73   -6.390e-02  4.595e-01  -0.139 0.889401
## factor(V7)A74   -9.178e-01  5.036e-01  -1.822 0.068385 .
## factor(V7)A75   -5.451e-02  4.701e-01  -0.116 0.907692
## V8               2.564e-01  1.020e-01   2.514 0.011923 *
## factor(V9)A92    1.914e-01  4.544e-01   0.421 0.673615
## factor(V9)A93   -4.284e-01  4.493e-01  -0.954 0.340301
## factor(V9)A94    1.025e-01  5.179e-01   0.198 0.843051
## factor(V10)A102  7.274e-01  4.829e-01   1.506 0.132021
## factor(V10)A103 -1.381e+00  5.262e-01  -2.624 0.008694 **
## V11              7.530e-03  9.910e-02   0.076 0.939429
## factor(V12)A122  3.813e-01  2.896e-01   1.317 0.187982
## factor(V12)A123  6.506e-02  2.754e-01   0.236 0.813246
## factor(V12)A124  6.558e-01  5.237e-01   1.252 0.210479
## V13             -1.642e-02  1.049e-02  -1.565 0.117581
## factor(V14)A142  1.555e-01  4.933e-01   0.315 0.752540
## factor(V14)A143 -7.023e-01  2.638e-01  -2.662 0.007770 **
## factor(V15)A152 -4.037e-01  2.643e-01  -1.528 0.126622
## factor(V15)A153 -6.057e-01  5.802e-01  -1.044 0.296522
## V16              2.693e-01  2.262e-01   1.190 0.233971
## factor(V17)A172  1.865e-01  7.735e-01   0.241 0.809465
## factor(V17)A173  2.613e-01  7.430e-01   0.352 0.725062
## factor(V17)A174  7.397e-02  7.433e-01   0.100 0.920728
## V18              5.661e-02  2.866e-01   0.198 0.843409
## factor(V19)A192 -9.199e-02  2.321e-01  -0.396 0.691788
## factor(V20)A202 -1.069e+00  7.352e-01  -1.454 0.145817
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 975.68  on 799  degrees of freedom
## Residual deviance: 692.88  on 751  degrees of freedom
## AIC: 790.88
##
## Number of Fisher Scoring iterations: 5

#cv.glm(train, model, K = 10)$delta #<---Cannot get this to work in R
Markdown. Output should be
#[0.1709936 0.1696361]
```

From here we will use the test data set to determine the optimum threshold for minimizing our cost function. It is 5 times worse to flag a bad customer as good than it is to flag a good customer as bad. In our transformed data set, 0 denotes a good customer and 1 denotes a bad customer. Due to this our cost function will be 5*FN + 1*FP where FN is a False Negative and FP is a False Positive.

```
set.seed(1)

pred = predict.glm(model, newdata = test, type = 'response')

thresholdlist <- c(1:50)/50
totalcost = matrix(, length(thresholdlist), ncol = 2)
n = 1
for (testthresh in thresholdlist) {
  answerx = matrix(, nrow(test), ncol = 1)
  for (x in 1:length(pred)) {
    if (pred[x] >= testthresh) {
      answerx[x] = 1
    } else
      answerx[x] = 0
  }
  cm = confusionMatrix(data = as.factor(answerx),
                       reference = as.factor(test$V21),
                       positive = '0')
  cost = cm$table[2, 1] * 1 + cm$table[1, 2] * 5
  totalcost[n, 1] = testthresh
  totalcost[n, 2] = cost
  n = n + 1
}

## Warning in confusionMatrix.default(data = as.factor(answerx), reference =
## as.factor(test$V21), : Levels are not in the same order for reference and
## data. Refactoring data to match.

## Warning in confusionMatrix.default(data = as.factor(answerx), reference =
```

```
## as.factor(test$V21), : Levels are not in the same order for reference and
## data. Refactoring data to match.

opt_threshold <- totalcost[which.min(totalcost[, 2]), 1]

answerx = matrix(, nrow(test), ncol = 1)
for (x in 1:length(pred)) {
  if (pred[x] >= opt_threshold) {
    answerx[x] = 1
  } else
    answerx[x] = 0
}

cm = confusionMatrix(data = as.factor(answerx),
                     reference = as.factor(test$V21),
                     positive = '0')
cm

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 42  4
##          1 97 57
##
##                Accuracy : 0.495
##                  95% CI : (0.4237, 0.5664)
##     No Information Rate : 0.695
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1657
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.3022
##             Specificity : 0.9344
##          Pos Pred Value : 0.9130
##          Neg Pred Value : 0.3701
##              Prevalence : 0.6950
##          Detection Rate : 0.2100
##    Detection Prevalence : 0.2300
##       Balanced Accuracy : 0.6183
##
##        'Positive' Class : 0
##

## Optimum threshold is 0.06
```

From the above we see that the optimum threshold is at 0.06 and that this threshold leads to 4 FNs and 97 FPs for the test data set.