Predicting the 2016 US Presidential Election

Objective

The purpose of this assignment was to utilize both the decision tree and k-nearest neighbor algorithms to retroactively predict the winner of the 2016 US Presidential Election.

The Data Set

The analysis is based on Professor Allan Lichtman's "Keys to the White House", a thirteen question model of predicting the presidential election based on the performance of the party currently occupying the White House. The data set is composed of a modified twelve questions for every election from 1860 to 1980. The twelve true or false questions are listed below:

1. Has the presidential party been in power for more than one term?
2. Did the presidential party receive more than 50% of the popular vote in the last election?
3. Was there significant activity of a third party during the election year?
4. Was there serious competition in the presidential party primaries?
5. Was the presidential party candidate the president at the time of the election?
6. Was there a depression or recession in the election year?
7. Was there a growth in the GDP of more than 2.1% in the year of the election?
8. Did the presidential party president make any substantial political changes during his term?
9. Did significant social tension exist during the term of the presidential party?
10. Was the presidential party administration guilty of any serious mistakes or scandals?
11. Was the presidential party candidate a national hero?
12. Was the opposition party candidate a national hero?

The resulting data set includes a limited 30 instances with 13 associated attributes including the winner classification. Of the 30 instances, 17 of the election winners, or 56.67% percent, were from the party occupying the White House with the remaining 13 winners, or 43.33%, coming from the opposition party.

```r
#import data set and display structure
elect <- read.csv("elect.csv", header=TRUE)
str(elect)
```

```
## 'data.frame':    30 obs. of  13 variables:
##  $ Winner: Factor w/ 2 levels "O","P": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Q1    : int  0 1 1 0 0 1 1 0 0 1 ...
##  $ Q2    : int  0 1 0 0 1 1 1 0 1 1 ...
##  $ Q3    : int  0 0 0 0 0 0 0 0 1 0 ...
##  $ Q4    : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ Q5    : int  1 1 0 1 1 1 0 1 1 0 ...
##  $ Q6    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Q7    : int  0 1 1 0 1 0 0 0 1 1 ...
##  $ Q8    : int  1 0 1 0 0 0 1 1 1 0 ...
##  $ Q9    : int  1 0 0 0 0 0 0 0 0 0 ...
##  $ Q10   : int  0 0 0 0 0 0 0 0 1 0 ...
##  $ Q11   : int  0 1 0 0 0 1 0 0 0 0 ...
##  $ Q12   : int  0 0 0 0 1 0 1 0 0 0 ...
```

Analysis

Training and Test Sets

Given the small sample size, the random testing sets were comprised of 5 instances, including 3 from the presidential party and 2 from the opposition party winners. This split maintains a 60%/40% ratio between the party designations which is as close to the ratio of the full data set as possible. Eight testing sets were randomly sampled and combined into one table in the code below.

```
#randomly sampled line numbers for each test set
test_set1 <- c(3, 8, 16, 22, 28)
test_set2 <- c(2, 14, 17, 20, 22)
test_set3 <- c(2, 10, 17, 24, 26)
test_set4 <- c(6, 12, 17, 24, 28)
test_set5 <- c(1, 5,  9, 20, 26)
test_set6 <- c(3, 7, 16, 20, 29)
test_set7 <- c(5, 12, 13, 26, 30)
test_set8 <- c(10, 13, 14, 27, 28)

#combine all test sets into one
test <- cbind(test_set1, test_set2, test_set3, test_set4, test_set5, test_set6, test_set7, test_set8)
test
```

```
##      test_set1 test_set2 test_set3 test_set4 test_set5 test_set6 test_set7
## [1,]         3         2         2         6         1         3         5
## [2,]         8        14        10        12         5         7        12
## [3,]        16        17        17        17         9        16        13
## [4,]        22        20        24        24        20        20        26
## [5,]        28        22        26        28        26        29        30
##      test_set8
## [1,]        10
## [2,]        13
## [3,]        14
## [4,]        27
## [5,]        28
```

Building the Tree

Using the R package and function rpart, each of the sample training sets were used to build a decision tree model. The code, which returns the prediction error rates for each of the test sets at the specified minimum split value, is shown below.

```
build_tree <- function(data, testset, minsp, splt) {
  #Input  - data is the full data set
  #       - testset is the random sample line numbers of data set
  #         or separate test set
  #       - minsp is the minimum split for the tree
  #       - splt is a boolean operator describing if the training and
  #         test sets are split within the same data set
  #Output - array of prediction error rates for each of the test sets

  if (splt == TRUE) {
    elect.test <- data[testset,]
    elect.train <- data[-testset,]
  } else {
    elect.test <- testset
    elect.train <- data
  }

  #build the decision tree with the training set and associated minimum split value
    tree <- rpart(Winner ~ ., method="class", data=elect.train,
                  control=rpart.control(minsplit=minsp))

  #use fitted decision tree model to predict results of the testing set
    tree.pred <- predict(tree, elect.test, type="class")

  #calculate prediction accuracy and associated error rates
    Accuracy <- sum(tree.pred == elect.test$Winner)/nrow(elect.test)
    Error <- 1 - Accuracy
    final <- rbind(Accuracy, Error)
    return(Error)
}
```

Minimum Split = 2

For the original set of models, the minimum split, or the value needed for the branch of the tree to split, was set to 2 as this would provide the most opportunities for the tree to grow.

```
#set number of test sets
n = ncol(test)

#initialize array for results of the decision trees
result_dt = array(0, dim=c(4,n))
rownames(result_dt) <- c("Min Split = 2", "Min Split = 3",
                         "Min Split = 4", "Min Split = 5")

#build and predict models for each of the random test sets at
#minimum splits from 2 to 5
for (j in c(2:5)) {
  for (i in c(1:n)) {
    result_dt[j-1,i] <- build_tree(elect, test[,i], j,TRUE)
  }
}

#append array of error rates with the row average
result_dt <- cbind(result_dt, rowMeans(result_dt))
colnames(result_dt) <- c("Test 1", "Test 2", "Test 3", "Test 4",
                         "Test 5", "Test 6", "Test 7","Test 8", "Average")

#extract prediction errors including minimum split of 2
result_dt[1,]
```

```
##  Test 1  Test 2  Test 3  Test 4  Test 5  Test 6  Test 7  Test 8 Average
##   0.200   0.000   0.000   0.400   0.600   0.400   0.200   0.400   0.275
```

As can be seen above, there is a wide range of prediction error rates culminating in an average error rate of 27.5% for decision trees with a minimum split of 2.

Minimum Split = 3

```r
#extract prediction errors including minimum split of 3
result_dt[1:2,]
```

```
##                Test 1 Test 2 Test 3 Test 4 Test 5 Test 6 Test 7 Test 8
## Min Split = 2    0.2      0      0    0.4    0.6    0.4    0.2    0.4
## Min Split = 3    0.2      0      0    0.0    0.6    0.4    0.2    0.4
##                Average
## Min Split = 2   0.275
## Min Split = 3   0.225
```

Increasing the minimum split from 2 to 3 proceeded to decrease the average error by 5%. This decrease was the result of increased accuracy of 40% in test set 4 with the rest of the test sets maintaining the same error rate.

Minimum Split = 4

```r
#extract prediction errors including minimum split of 4
result_dt[1:3,]
```

```
##                Test 1 Test 2 Test 3 Test 4 Test 5 Test 6 Test 7 Test 8
## Min Split = 2    0.2      0      0    0.4    0.6    0.4    0.2    0.4
## Min Split = 3    0.2      0      0    0.0    0.6    0.4    0.2    0.4
## Min Split = 4    0.2      0      0    0.0    0.6    0.4    0.2    0.2
##                Average
## Min Split = 2   0.275
## Min Split = 3   0.225
## Min Split = 4   0.200
```

Once again, the average error rate decreased when increasing the minimum split from 3 to 4. This decrease, while significantly less than the first decrease, was the result of increased accuracy of 20% in just test set 8.

Minimum Split = 5

```r
#extract prediction errors including minimum split of 5
result_dt
```

```
##                Test 1 Test 2 Test 3 Test 4 Test 5 Test 6 Test 7 Test 8
## Min Split = 2    0.2      0      0    0.4    0.6    0.4    0.2    0.4
## Min Split = 3    0.2      0      0    0.0    0.6    0.4    0.2    0.4
## Min Split = 4    0.2      0      0    0.0    0.6    0.4    0.2    0.2
## Min Split = 5    0.2      0      0    0.0    0.2    0.4    0.2    0.2
##                Average
## Min Split = 2   0.275
## Min Split = 3   0.225
## Min Split = 4   0.200
## Min Split = 5   0.150
```

For the third and final increase in the minimum split from 4 to 5, the average error rate decreased once more by 5%. This decrease was the result of increased accuracy of 40% in just test set 5. This means that for a minimum split of 5, there is an average total accuracy of 85%.

kNN

In comparison, the R package Class with the function knn, was used to test both the 1NN and 3NN classification rules on the same randomly selected test sets.

```r
#initialize array of resulting error rates for knn predictions
result_knn = array(0, dim=c(2,n))
rownames(result_knn) <- c("kNN k=1","kNN k=3")

kNN_prediction <- function(data, testset, k, splt) {
  #Input  - data is the full data set
  #       - testset is the random sample line numbers of data set
  #         or separate test set
  #       - k is the value of k
  #       - splt is a boolean operator describing if the training and
  #         test sets are split within the same data set
  #Output - array of prediction error rates for each of the test sets

  #remove winner classification from data set
  elect.n <- data[,-1]

  #set testing and training sets depending on value of splt
  if (splt == TRUE) {
      elect.train <- elect.n[-testset, ]
      elect.test <- elect.n[testset, ]
      elect.train.labels <- data[-testset,1]
      elect.test.labels <- data[testset,1]
  } else {
    elect.train <- elect[,-1]
    elect.test <- testset[,-1]
    elect.train.labels <- elect[,1]
    elect.test.labels <- testset[,1]
  }

    #build model using training set and predict accuracy for testing set
  elect.test.pred <- knn(train = elect.train, test = elect.test,
                         cl = elect.train.labels, k=k)

  #calculate the accuracy and error rates for the prediction results
    Accuracy <- (sum(elect.test.pred == elect.test.labels)/nrow(elect.test))
    Error <- (1-Accuracy)
    final <- rbind(Accuracy,Error)
    return(Error)
}

#vector of k values
K = c(1,3)

#build and predict models for each of the random test sets at k = 1 and k = 3
for (j in range(1,2)) {
  for (i in c(1:n)) {
    result_knn[j,i] <- kNN_prediction(elect,test[,i],K[j],TRUE)
  }
}

#append array of error rates with the row average
result_knn <- cbind(result_knn, rowMeans(result_knn))
colnames(result_knn) <- c("Test 1", "Test 2", "Test 3", "Test 4",
                             "Test 5", "Test 6", "Test 7","Test 8", "Average")

#display prediction errors for kNN at both k=1 and k=3
result_knn
```

```
##         Test 1 Test 2 Test 3 Test 4 Test 5 Test 6 Test 7 Test 8 Average
## kNN k=1    0.2    0.0    0.4    0.2    0.2    0.4    0.2    0.2   0.225
## kNN k=3    0.2    0.2    0.0    0.0    0.0    0.2    0.0    0.0   0.075
```

For the method of k-nearest neighbor alone, 3NN is the better option as it decreases the error rate by 15% when compared to 1NN, increasing the total accuracy to 92.5%.

Decision Trees vs kNN

```
#append knn results array with the decision trees array
result <- as.data.frame(rbind(result_dt, result_knn))

#sort table by descending average prediction errors
result[order(-result$Average),]
```

```
##               Test 1 Test 2 Test 3 Test 4 Test 5 Test 6 Test 7 Test 8
## Min Split = 2    0.2    0.0    0.0    0.4    0.6    0.4    0.2    0.4
## Min Split = 3    0.2    0.0    0.0    0.0    0.6    0.4    0.2    0.4
## kNN k=1          0.2    0.0    0.4    0.2    0.2    0.4    0.2    0.2
## Min Split = 4    0.2    0.0    0.0    0.0    0.6    0.4    0.2    0.2
## Min Split = 5    0.2    0.0    0.0    0.0    0.2    0.4    0.2    0.2
## kNN k=3          0.2    0.2    0.0    0.0    0.0    0.2    0.0    0.0
##               Average
## Min Split = 2   0.275
## Min Split = 3   0.225
## kNN k=1         0.225
## Min Split = 4   0.200
## Min Split = 5   0.150
## kNN k=3         0.075
```

However, in comparison to the decision tree prediction errors, 1NN is tied with minimum split = 3 for second worst, only better than average error when the minimum split is 2. Therefore, there are still two decision trees that are better than 1NN, yet both are still eclipsed by 3NN which has half the error rate of the next best option at minimum split = 5.

In addition, based on the training sets, the decision trees were built by calculating the entropy of questions 1 through 12 which were then compared to the entropy of the presidential party. The question resulting in the largest difference, or the largest information gain, becomes a node on the tree before the process is repeated until the minimum split level has been reached. This means that not all questions were included in the decision trees, as well as some never being included, and therefore limits the scope of the overall data.

K-nearest neighbor is the better prediction option as this method takes all questions/predictors into account and not just the ones entropy and information gain deem most useful which has clearly resulted in the highest accuracy. Therefore, I believe that 3NN is the best method to be used for prediction.

Prediction

As determined previously, the tree models with a minimum split of 5 are the most accurate of the decision trees while the 3NN model is the most accurate in terms of just kNN as well as for both algorithms. Therefore, those two models are used for the prediction of the result of the 2016 US Presidential election.

Training and Testing Sets

The entire data set of all previous elections was used as the training set and the test set for the 2016 election was completed using my knowledge as well as information presented in an interview with Allan Lichtman conducted by the Washington Post in September. Lichtman describes his prediction process as follows:

"The keys are 13 true/false questions, where an answer of"true" always favors the reelection of the party holding the White House, in this case the Democrats. . . And if six or more of the 13 keys are false — that is, they go against the party in power — they lose. If fewer than six are false, the party in power gets four more years" (Washington Post).

At that point in September, Lichtman believed that there were five keys that were in favor of Trump as well as a wild card in the form of third party candidate, Gary Johnson. Lichtman stated, "One more key and the Democrats are down, and we have the Gary Johnson Key. One of my keys would be that the party in power gets a"false" if a third-party candidate is anticipated to get 5 percent of the vote or more" (Washington Post).

Lichtman's five keys not in the Democrats favor:

- "Key 1 is the party mandate — how well they did in the midterms. They got crushed. (different variation was used in our questions)
- Key number 3 is, the sitting president is not running.
- Key number 7, no major policy change in Obama's second term like the Affordable Car Act.
- Key number 11, no major smashing foreign policy success. (not included in our questions)
- And Key number 12, Hillary Clinton is not a Franklin Roosevelt" (Washington Post).

As he also believed that the third party candidate was a wild card, I concluded that the answers to the rest of the questions fell in favor of the Democrats and compiled my test sets accordingly.

Question Uncertainty

For the most part, there were only a few questions that I was uncertain about, specifically questions 1, 3 and 4.

1. Has the P-party been in power for more than one term?

- Question 1 is slightly different from Lichtman's Key 1 which takes into account the midterm elections and the party in power in Congress and not just the party currently in the White House. To see if this was a significant distinction, test sets with both yes and no were included.

3. Was there significant activity of a third party during the election year?

- I personally do not believe that there was significant activity of a third party this election, as no third party candidate has garnered an electoral vote or 5% of the popular vote since 1996; however, Lichtman believed Gary Johnson had the potential to pull in 5% of the popular vote, so I included versions with question 3 as both yes and no. (Washington Post)

4. Was there serious competition in the P-party primaries?

- Regarding Question 4, it can be said that there was or was not serious competition in the presidential party primaries. Therefore, I also included versions with both yes and no in the test sets.

Therefore, the models fitted with the full data set are then used to predict the results on six different test sets taking into account all variations of question uncertainty.

```r
#importing full data set appended to include the six test sets for the 2016 election
pred_elect <- read.csv("pred_elect.csv", header=TRUE)
```

Below is the code building and predicting the models for each of the election test sets using both the decision tree and k-nearest neighbor algorithms.

```r
#initialize array of resulting error rates for 2016 decision tree predictions
pred_result_dt <- array(0, dim=c(1,6))

#initialize array of resulting error rates for 2016 knn predictions
pred_result_knn <- array(0, dim=c(1,6))

N <- c(31:nrow(pred_elect))
for (i in c(1:length(N))) {
  pred_result_dt[1,i] <- build_tree(elect, pred_elect[N[i],], 5, FALSE)
}

for (i in c(1:6)) {
  pred_result_knn[1,i] <- kNN_prediction(elect, pred_elect[N[i],], 3, FALSE)
}

#append array of decision tree error rate with array of knn error rates
pred_result <- rbind(pred_result_dt, pred_result_knn)

#append array of error rates with the row average
pred_result <- cbind(pred_result, rowMeans(pred_result))
rownames(pred_result) <- c("Min Split = 5", "kNN k=3")
colnames(pred_result) <- c("Test 1", "Test 2", "Test 3", "Test 4",
                           "Test 5", "Test 6", "Average")

#print result
pred_result
```

```
##               Test 1 Test 2 Test 3 Test 4 Test 5 Test 6   Average
## Min Split = 5      1      0      1      0      1      0 0.5000000
## kNN k=3            1      1      1      1      1      0 0.8333333
```

Results

- Test Set 1 included my initial decisions regarding the prediction questions, ignoring the midterm elections (Q1) as well as third party candidates (Q3) and competition in the primaries (Q4). I have used this set as the baseline for the other versions. None of the methods resulted in an accurate prediction.
- For Test Set 2, the only change from Test Set 1 was answering yes for serious competition in the p-party primaries (Q4). This proved to be a determining factor in the decision tree prediction as Q4 is the root node, but was not significant for the kNN method.
- Test Set 3 only differed from Test Set 1 by taking the results of the midterm elections into account (Q1=n). The results reaffirmed that the difference to Lichtman's Key 1 was not a significant factor in either method.
- Test Set 4 differed from Test Set 1 for midterm elections (Q1=n) and competition in the primaries (Q4=y). Once again, as Q4 is the root node of the tree, an affirmative answer was significant for the decision tree prediction, but not for 3NN.
- Test Set 5 differed from Test Set 1 with an answer of yes to significant third party activity (Q3), however this change alone proved insignificant across both methods.
- Test Set 6 included answers of yes for both third party (Q3) as well as primary competition (Q4), with all other answers remaining the same as Test Set 1. This combination was the only version that correctly predicted Trump in both methods.

Conclusion

The American electoral system is very complex with the electoral college electing the president, not the American population, and thus can produce a paradox where the electoral vote and the popular vote contradict.

Though this is uncommon, the 2016 election was the second occurrence in the last 16 years, the same situation occurring in 2000 when Gore won the popular vote, but Bush was elected president.

Therefore, the results were technically accurate if we were attempting to predict the popular vote as Hillary Clinton and the P-Party did in fact win by around 3 million votes or just over 2%; however losing the electoral college vote lost her the election. (United States presidential election, 2016)

To avoid this paradox, it would be more appropriate and accurate to predict the electoral vote and the popular vote separately as the electoral vote cannot be predicted nationally, but state by state.

In conclusion, 3NN proved to be the most accurate method when testing samples of the data set with an error rate of just 7.5%. If we were predicting the winner of the popular vote for the 2016 election, 3NN again proved to be the most accurate method with an average error rate of 16% for all test sets.

- In addition, as no third party candidate managed to pull in 5% of the popular vote, it would be reasonable to remove test sets 5 and 6 from consideration as question 3 should not have been a factor which would then result in 100% accuracy for 3NN across the remaining four test sets. (List of third party performances in United States presidential elections)

However, if we were predicting the actual winner of the election, none of these methods prove to be worthwhile or useful in predicting an accurate outcome given that the average error rate was, at best, 50%.