2008 Democratic Party Presidential Primaries Analysis and Classification Case study

Huseyin Can Minareci & Noam Shmuel

June, 2020

Contents

Introduction

In 2008 the Democratic Party chose its nominee for President of the United States for the presidential election which took place later on that year. The two nominees, Senator Hillary Clinton and Senator Barack Obama, had a close race during the contest. In this report we aim to find the best classification model to predict who will come out as a winner (dependent vatiable) in each county (observation).

Data preparation

The data is comprised of 2450 observations (counties) and 50 variabels, where the variables are:

`winner` - The winner in the county, Obama or Clinton. This is the dependent variable in all of the upcoming models.
`POP05_SQMI`- The 2005 estimated population of the ZIP Code area per square mile
`tvotes` - total votes in the obsereved county
`popUnder30_00` - precent of population under the age of 30 in the observed country
`pop65up_00` - precent of population over the age of 65 in the observed country
`presVote04` - total votes in the 2004 presidential election
`kerry04` - rate of population voted for John Kerry in 2004 presidential elections
`Bush04` - rate of population voted for George W. Bush in 2004 presidential elections
`pres04margin` - rate of margin won by George W. Bush in 2004 presidential elections
`pres04winner` - 2004 presidential elections winner in the obsereved county
`pop06` - total population in the observed country
`pct_less_30k` - rate of population earning less than 30K annually in the observed country
`pct_more_100k` - rate of population earning more than 100K annually in the observed country
`pct_hs_grad` - rate of population with highschool diploma in the observed country
`pct_labor_force` - rate of population perticipating in the labor force
`pct_homeowner` - rate of home owners in the observed country
`unempFeb07` - unemployment rate in Februar 2007 (prior to the 2008 economic crisis)
`unempFeb08` - unemployment rate in Februar 2008
`unempChg` - unemployment change between Februar 2008 to Februar 2009
`poverty05` - poverty rate in 2005
`median_hhi05` - Median household income in 2005
`Catholic` - rate of Catholic in the observed country
`So.Bapt.Conv` - rate of Baptists in the observed country
`Un.Methodist` - rate of Methodist in the observed country
`Construction` - precent of population working in Construction
`Manufacturing` - precent of population working in Manufacturing
`FinancialActivities` - precent of population working in Financial Activities
`GoodsProducing` - precent of population working in Goods Producing
`ServiceProviding` - precent of population working in ServiceProviding

Moreover, there are a few variables which are presented as nominal.
Variables such as: `white06`, `black06`, `indian06`, `asian06`, `hawaii06`, `mixed06` will be normilized against the population in 2006 (`pop06`) in order to represent the ratio of these ethnicity groups in the observed county. In addition, instead of showing both the population in the years 2006 and in 2000, a precent change will be more appropriate:

```
raw.data <-
(
  raw.data %>%
    mutate(pct_white06 =  white06/pop06) %>%
    mutate(pct_black06 =  black06/pop06) %>%
    mutate(pct_indian06 =  indian06/pop06) %>%
    mutate(pct_asian06 =  asian06/pop06) %>%
    mutate(pct_hawaii06 =  hawaii06/pop06) %>%
    mutate(pct_mixed06 =  mixed06/pop06) %>%
    mutate(pct_change_06_00 =  ((pop06/pop00 -1)*100) )
)
```

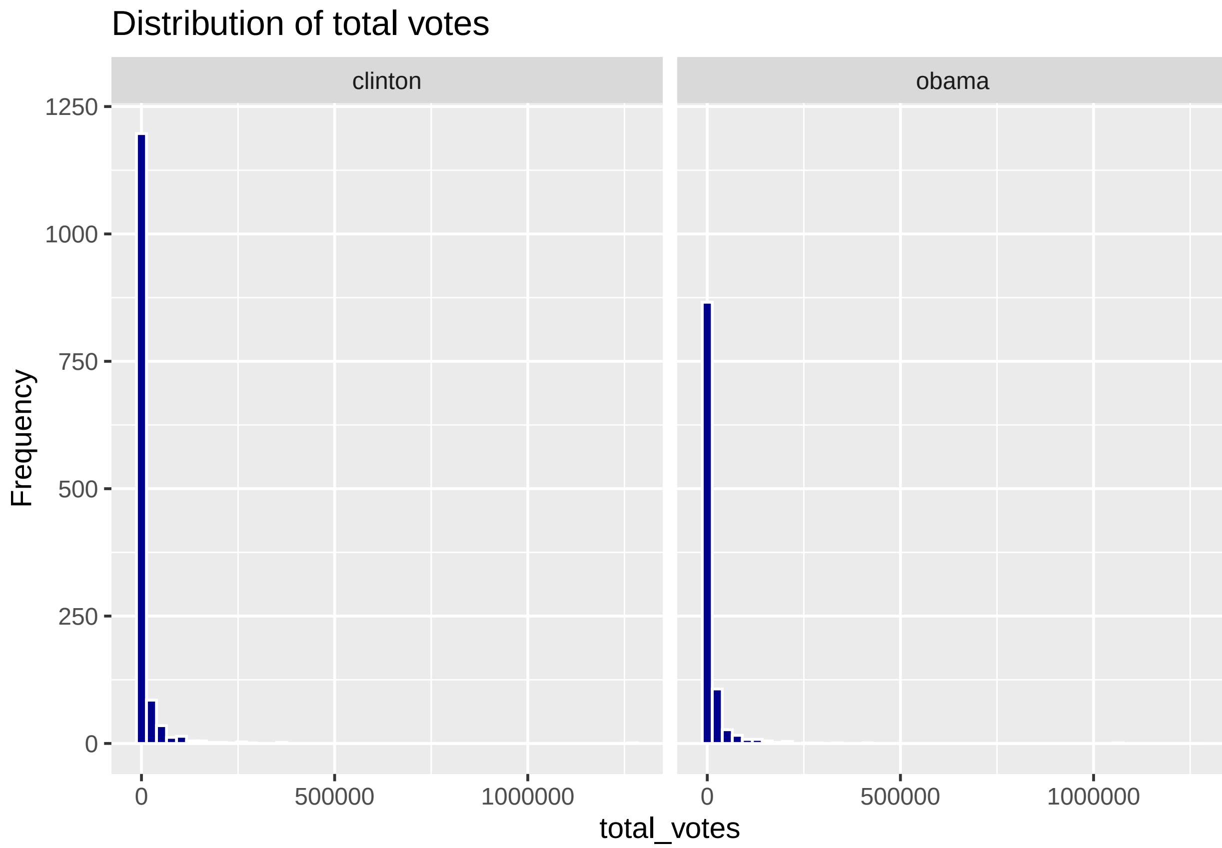
After the creating the variables above and rearranging them, the dataset looks like:

winner	POP05_SQMI	tvotes	popUnder30__00	pop65up__00	presVote04	kerry04	Bush04	pres04margin	pr
obama	78.9	4118	30.3	10.2	20081	0.24	0.76	0.52	bu
clinton	97	12085	27.2	15.5	69320	0.23	0.76	0.54	bu
obama	32.3	3823	31.4	13.3	10777	0.45	0.55	0.1	bu
clinton	33.4	1751	32.8	11.6	7600	0.27	0.72	0.45	bu
clinton	80.7	3471	30.2	12.9	21504	0.18	0.81	0.63	bu
NA	...	...	...	...	...	...	...	...	N
clinton	3.6	596	29.7	8	16272	0.32	0.65	0.33	bu
obama	4.5	1150	38.6	6.9	11359	0.53	0.45	-0.07	ke
obama	9.6	168	28.5	7	8081	0.22	0.75	0.53	bu
obama	3.6	98	23.6	15.9	4114	0.21	0.78	0.57	bu
clinton	2.8	68	24.7	15.6	3392	0.17	0.81	0.64	bu

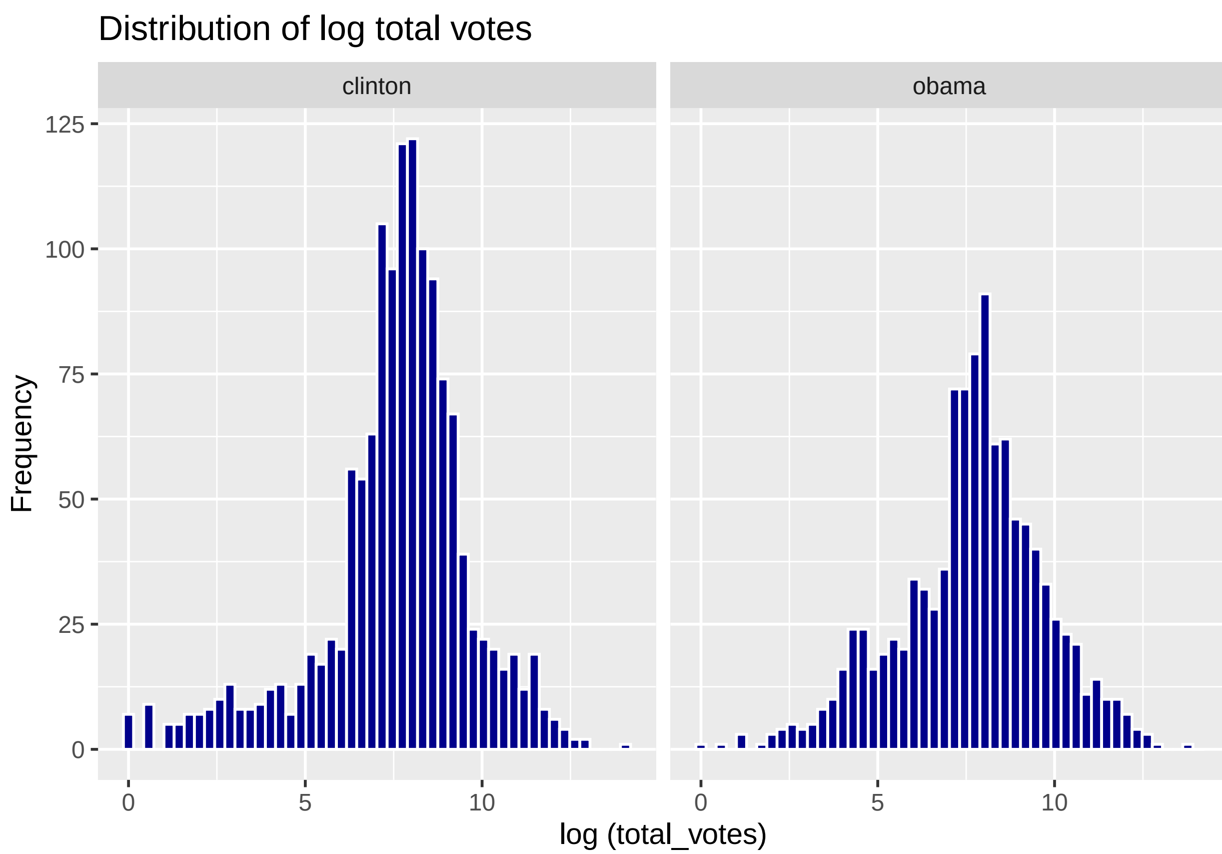
Now that the dateset it arranged in proper way, the next step would be to get to know the data with explanatory data analysis:

Explanatory Data Analysis and Summary Statistics

Next step would be to observing key independent varibles and their distribution. It's crystal clear that the total votes, `tvotes`, is skewed.



Therefor, we will perform a log transformation on this vatiable to make it normally distributed:



Next step would be a summary statistics overview of the independent variables:

	vars	n	mean	sd	median	trimmed	mad	min	max	r
POP05_SQMI	1	2449	282.70	1744.86	44.60	76.71	47.44	0.10	57173.00	5
tvotes	2	2450	11792.21	45099.70	2469.50	4002.93	3111.98	1.00	1271094.00	1
popUnder30_00	3	2449	28.95	4.98	28.60	28.60	3.71	11.90	60.00	4
pop65up_00	4	2449	14.42	4.00	14.10	14.25	3.56	1.80	34.70	3
presVote04	5	2449	45242.98	125747.06	11698.00	19559.93	11547.97	80.00	3023280.00	3
kerry04	6	2449	0.39	0.13	0.39	0.39	0.13	0.07	0.89	0
Bush04	7	2449	0.60	0.13	0.60	0.60	0.13	0.09	0.92	0
pres04margin	8	2449	0.20	0.25	0.21	0.21	0.25	-0.80	0.85	1
pop06	9	2449	113964.18	359707.48	28785.00	47072.90	28818.78	60.00	9948081.00	9
pct_change_06_00	10	2449	3.95	9.10	2.48	2.95	6.16	-76.85	64.24	1
pct_white06	11	2449	0.76	0.20	0.81	0.79	0.19	0.02	0.99	0
pct_black06	12	2449	0.10	0.15	0.03	0.07	0.04	0.00	0.85	0
pct_indian06	13	2449	0.01	0.04	0.00	0.00	0.00	0.00	0.79	0
pct_asian06	14	2449	0.01	0.02	0.00	0.01	0.00	0.00	0.32	0
pct_hawaii06	15	2449	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0
pct_mixed06	16	2449	0.01	0.01	0.01	0.01	0.00	0.00	0.11	0
pct_less_30k	17	2448	0.43	0.11	0.44	0.43	0.10	0.08	0.74	0
pct_more_100k	18	2448	0.07	0.05	0.05	0.06	0.02	0.00	0.38	0
pct_hs_grad	19	2448	0.77	0.09	0.78	0.77	0.09	0.35	0.97	0
pct_labor_force	20	2448	0.60	0.07	0.61	0.61	0.07	0.32	0.86	0
pct_homeowner	21	2448	0.74	0.08	0.75	0.75	0.06	0.20	0.90	0
unempFeb07	22	2441	5.57	2.13	5.10	5.33	1.78	1.70	21.90	2
unempFeb08	23	2441	5.70	2.19	5.30	5.46	1.93	1.60	23.20	2
unempChg	24	2441	0.13	0.90	0.10	0.11	0.89	-3.80	9.30	1
poverty05	25	2450	15.70	6.63	14.80	15.16	6.23	2.50	46.40	4
median_hhi05	26	2450	39551.62	10571.43	37440.00	38290.28	8347.04	17843.00	98245.00	8
Catholic	27	2447	0.14	0.15	0.09	0.11	0.11	0.00	0.95	0
So.Bapt.Conv	28	2447	0.16	0.16	0.12	0.14	0.17	0.00	0.96	0
Un.Methodist	29	2447	0.06	0.04	0.05	0.06	0.04	0.00	0.34	0
Construction	30	2447	Inf	NaN	6.19	6.55	3.54	0.00	Inf	I
Manufacturing	31	2447	Inf	NaN	13.72	14.70	12.16	0.00	Inf	I
FinancialActivities	32	2449	Inf	NaN	4.65	4.82	1.69	0.00	Inf	I
GoodsProducing	33	2445	Inf	NaN	28.90	29.69	13.20	0.00	Inf	I
ServiceProviding	34	2448	Inf	NaN	71.01	70.15	13.22	0.00	Inf	I
log_tvotes	35	2450	7.58	2.13	7.81	7.72	1.56	0.00	14.06	1

It accures that the variables `Construction`, `Manufacturing`, `FinancialActivities`, `GoodsProducing`, `ServiceProviding` have some Infinite values. Let examine how many Infinite values are there under `Construction`, but sorting the values in descending order:

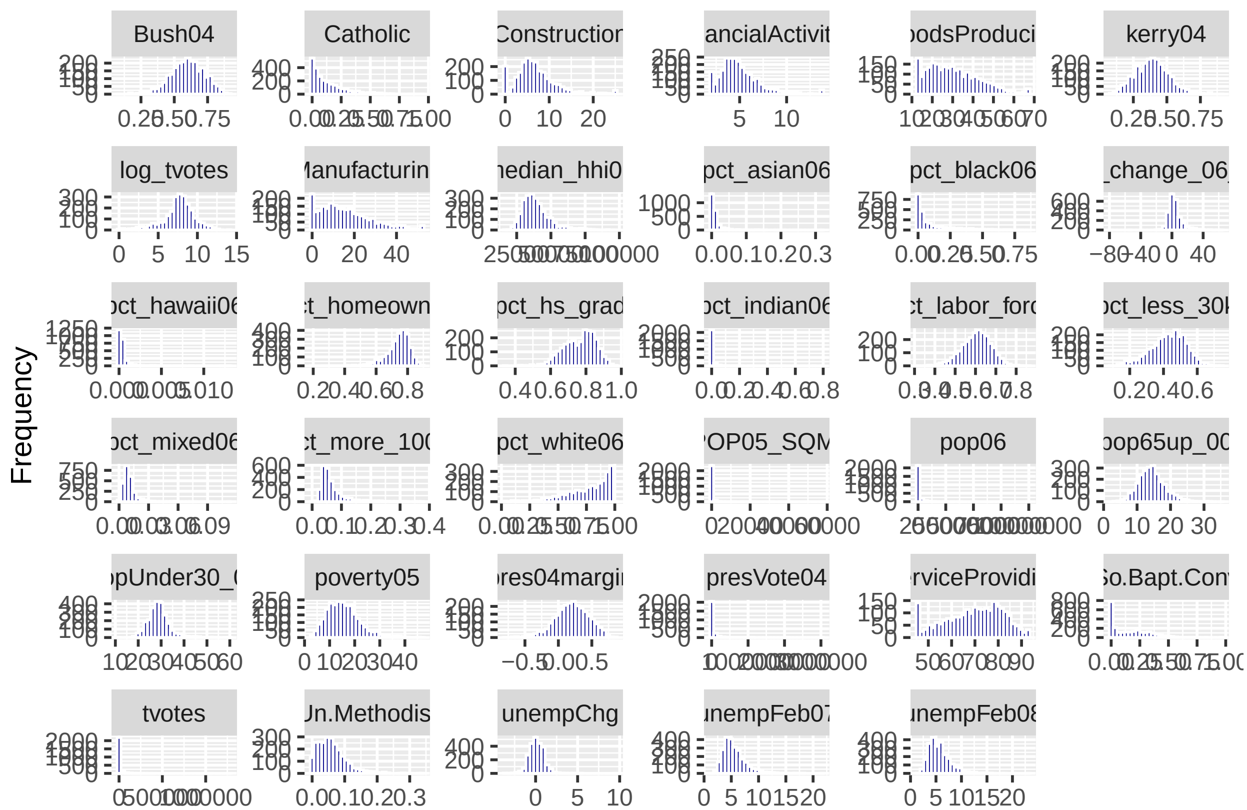
Construction
Inf
Inf
Inf
Inf
47.98992
39.46144
36.90827
35.70033

The next step would be handaling these infinite values. There is no absolute best partice with how to deal with Inf value, we could either remove these records, or replace them with other values, such as `0`. However we would use winsorization and replce them with the 99th quantile.

```r
pro_vec <- c("Construction", "Manufacturing", "FinancialActivities",
             "GoodsProducing", "ServiceProviding"  )
for (colname in pro_vec) {
  qua99 <- quantile(raw.data[[colname]],probs = 0.99,na.rm = T)
  vecc <- raw.data[[colname]]
  raw.data[[colname]] <- DescTools::Winsorize(vecc,maxval = qua99,na.rm = T)
}
```

```
## Registered S3 method overwritten by 'DescTools':
##   method         from
##   reorder.factor gdata
```

The dependent variables are distributed accordingly:



Missing Values

In the exploratory data analysis we have removed problematic values of `inf+`, we are going to apply similar and yet different aproach on the `NA` values in the independent variables.
However, First step would be the dependent variable, `winner`, and its distribution between the two candidates:

```r
table(raw.data$winner,  exclude = NULL)
```

```
##
## clinton   obama    <NA>
##    1369    1061      20
```

As shown in the table above, there are 20 missing values of the dependent variable,`winner`, therefore, these 20 observations will be removed from the dataset.

Moving to handling missing values in the independent variables.
The number of missing values:

```r
sum(colSums(is.na(raw.data)))
```

```
## [1] 73
```

There are 73 `NA`'s in total in the dataset, which consist at the most 3.004% of records.

As seen in the numeric variables distribution plot, most of the variables are distributed in normal mannor, therefore we can conclude that replacing missing values with the median of each variable would be good parctice. Replacing `NA`'s with median is presented in the function below

```r
for (col_name in raw.data %>%  keep(is.numeric) %>%   colnames()  ) {
   col_median <- median(raw.data[[col_name]], na.rm = TRUE)
   raw.data[[col_name]][is.na(raw.data[[col_name]])] <- col_median
}
rm(col_median,col_name)
```

Verifying that the missing values were indeed successfuly removed

```r
sum(colSums(is.na(raw.data %>%  keep(is.numeric))))
```

```
## [1] 0
```

Now that we have dealt with abnormal values, such as infinite, and missing values, the dataset is prepared for the main part of the analysis.

Data Partition

We start with partitioning the dataset in ratio 80-20 towards the train dataset, and making sure that the 80-20 ratio will be kept within the dependent varible in the train and test datasets.

```r
set.seed(123321)
training_index <- createDataPartition(primaries$winner, p = 0.8, list = FALSE)
primaries_train <- primaries[training_index,]
primaries_test <- primaries[-training_index,]
```

Verifying that the dependent variable is distributed in equal rates between the train and test datasets:

```r
prop.table(table(primaries_train$winner))
```

```
##
##  clinton    obama
## 0.563786 0.436214
```

```r
prop.table(table(primaries_test$winner))
```

```
##
##   clinton     obama
## 0.5628866 0.4371134
```

The partition of the dataset went well, the dependent variable is distributed evenly between the the train and test datasets.

Logistic Model

We start the classification part with Logistic Modeling.
In the first Model of the Logit we use every single one of the variables we have in the dataset as predictors:

```r
model1 <- glm(winner ~.,data=primaries_train, family=binomial(link = "logit"))
```

Just like suspected in the explanatory data analysis part, it seems like there are many varibles are collinear with one another. The output of the model reveals the variables such as `kerry04`, `Bush04` and `pres04margin` are insignificant most likely for the reason that they are correlated. Therefore we shall remove both `kerry04`and `Bush04`. The same with `unempFeb07`, `unempFeb08` and `unempChg`, we shall keep only the latter one.

```r
model2 <- glm(winner ~., family=binomial(link = "logit"),
              data=primaries_train %>%
                dplyr::select(c(-kerry04,-Bush04,-unempFeb08,-unempFeb07))
              )
```

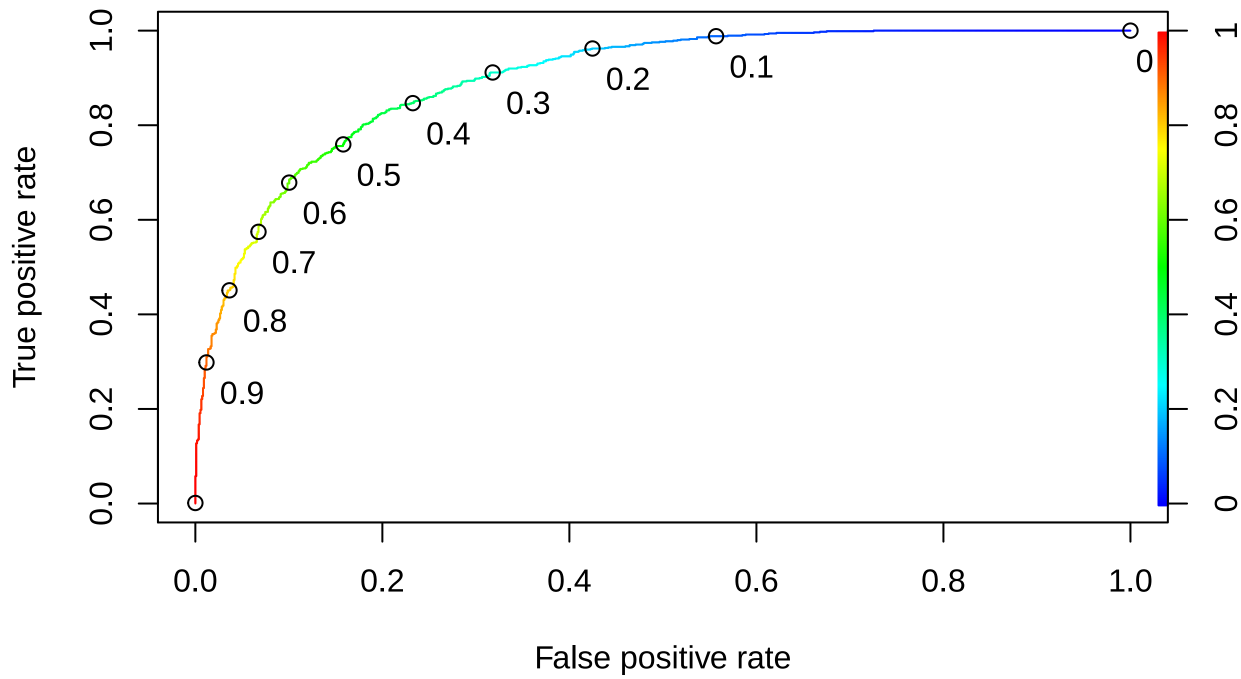
The model with all segnificant predictors are `model3`:

```r
model3 <- glm(winner ~., family=binomial(link = "logit"),
              data=primaries_train %>%
                dplyr::select(c(-kerry04,-Bush04,-unempFeb08,-unempFeb07,-pres04winner,
                         -pct_hawaii06,-pct_asian06,
                         -FinancialActivities,-Manufacturing,
                         -unempChg,
                         -pop65up_00,
                         -presVote04,
                         -pct_more_100k,
                         -median_hhi05,
                         -pct_white06,
                         -popUnder30_00))
              )
```

Our incentive of course would be predicting correctly which candidate each county vote for, however this task is less straight forward as our prediction for "success" is a probability between 0 and 1. As we set our threshold to be high we increase the number of false negatives- we predict Clinton and actully voted for Clinton **but** we decrease the number of false positives, counties which we predicted to vote for Obama, but actually voted for Clinton. On the other side, if we set the threshold to be low- we decrease the number of false negatives, we decrease wrong predictions of voting for Clinton when it was actually Obama, **but** we increases false positives, we predict Obama but it was actully Clinton. We Understand that picking the right threshold is a pure business decistion. One tool to help us cope with this problem is the ROC curve (receiver operating characteristic) below.

```r
model3_fitted <- predict(model3, type="response")
pred3.train.rocr <- prediction(model3_fitted, primaries_train$winner)
perf3.train.rocr <- performance(pred3.train.rocr, "tpr", "fpr")

plot(perf3.train.rocr, colorize=TRUE, print.cutoffs.at=seq(0,1,by=0.1), text.adj=c(-0.3,1.8))
```



We try to get a threshold which maximize the Y axis (True Positive Rate) and minimize the X axis (False Positive Rate). It seems like the best point would be 0.48. The table below is presenting the confusion matrix with threshold of 0.48:

```r
table(predictions = ifelse(model3_fitted > 0.48, "obama", "clinton"),
      actual = primaries_train$winner)
```

```
##             actual
## predictions clinton obama
##     clinton     911   184
##     obama       185   664
```

The rate of goodness of fit of the table above is the following:

```r
sum(diag(table(predictions = ifelse(model3_fitted > 0.48, "obama", "clinton"),
      actual = primaries_train$winner))/
      (sum(rowSums(table(predictions = ifelse(model3_fitted > 0.48, "obama", "clinton"),
      actual = primaries_train$winner)))))
```

```
## [1] 0.8101852
```

Another tool to help us assest how well our model fit the data it AUC, Area Under the Curve, of the ROC graph below. As long as the ROC graph is stratched to the top-left corner the better the fit, and simply the AUC is the area under it. When our model is base on pure random chance, the ROC graph will be straight line strached from the bottom left to the top right, and the area would be 0.5 (simply as the chance of flipping a coin). When we have the best model which predict perfectly the ROC will streached from the bottom left verticlly up all the way. In that option our AUC would be equal to 1. The AUC of our model, based on the train dataset is between 0.8 to 0.9 which considered to be "Excellent discrimination" acording to Hosmer & Lemeshow (2013). Applied logistic regression.

```r
as.numeric(performance(pred3.train.rocr, "auc")@y.values)
```

```
## [1] 0.8970947
```

Let's make the prediction on the tests dataset to make sure that the model is indeed fitting well not only on the train dataset:

```r
model3_fitted_test<- predict(model3, type="response", newdata = primaries_test)
pred3.test.rocr <- prediction(model3_fitted_test, primaries_test$winner)
perf3.test.rocr <- performance(pred3.test.rocr, "tpr", "fpr")
as.numeric(performance(pred3.test.rocr, "auc")@y.values)
```

```
## [1] 0.8875527
```

It looks like both test and train dataset get good score on the AUC test.

K Nearest Neighbor - KNN model

The second modeling would be *KNN*.
The initial step in KNN modeling is to normalize the dataset. We do that in order to make sure that each of the variables will have the same impact in our modeling. The idea behind it, is the way the distance is calculated - KNN algorithm will use each variable is a dimention of space, and different variables a scaled differently, for example "number of vote" which range between a dozen of votes, to thousands, and on the other side, the variable "precent white" is range between 0 and 1 (as it represent the rate of white individules in the observed county). Therefore the euclidean distance in these two variables for the same observation would yield totally different distance.
There are different way to normalize your data, we pick with normalizing it as standard normal distribusion where for each variable the mean equal to `0` with standard deviation of `1`:

```r
scaled_data <- primaries %>%
  keep(is.numeric)

scaled_data <- as.data.frame(scale(scaled_data,center = TRUE,scale = T))
scaled_data <- cbind(winner=primaries$winner, scaled_data)
```

We shall devide the scaled dataset into train and test:

```r
scaled_train <- scaled_data[training_index, ]
scaled_test <-  scaled_data[-training_index,]
```

The initial model consists of all the variables as predictors, with the default parameter `k`=5:

```
model1_knn5 <-
  caret::train(winner ~ .,
        data = scaled_train,
        method = "knn")
```

Below is a table of the correct classifications:

```
table(predictions = predict(model1_knn5,scaled_train), obsereved =  scaled_train$winner)
```

```
##            obsereved
## predictions clinton obama
##     clinton     970   160
##     obama       126   688
```

The ratio of goodness of fit for the training data is showing below:

```
sum(diag(table(predict(model1_knn5,scaled_train), scaled_train$winner))/(sum(rowSums(table(predict(model
```

```
## [1] 0.8528807
```

Next step would be to check what would be the optimal K which will maximize the correct classifications. We will do it using Cross validation process of 5 folds (the dataset will be divided for 5 sections, 4 will be train and 1 will be test, in a loop for all the five), each time for a different K, where K will be between 1 to 99 with jumps of 4:

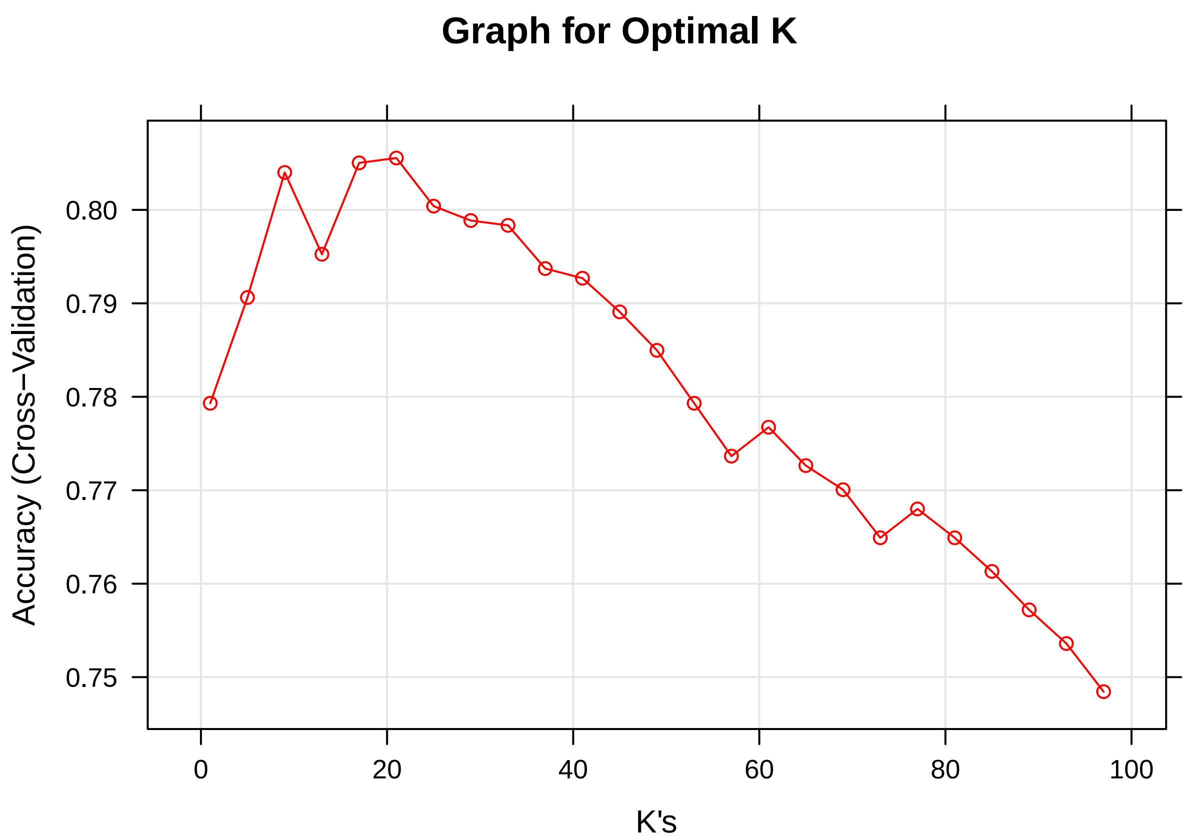
```
set.seed(223322)
different_k <- data.frame(k = seq(1, 99, 4))

trainControl_cv_6_folds <- trainControl(method = "cv", number = 6,classProbs = T)

model2_knn_n <-
  train(winner ~ .,
        data = scaled_train,
        method = "knn",
        trControl = trainControl_cv_6_folds,
        tuneGrid = different_k,
        metric = "ROC")
```

In the plot below is showing the correct classification ratio for each K:

```
plot(model2_knn_n,col = "red", main = "Graph for Optimal K", xlab= "K's")
```



From the graph above we can clearly see the `k` = 21 will maximize our model accuracy.

```
model2_knn_n$finalModel$k
```

```
## [1] 21
```

Below is a table of the correct classifications:

```
table(predictions = predict(model2_knn_n,scaled_train), obsereved =  scaled_train$winner)
```

```
##            obsereved
## predictions clinton obama
##     clinton     954   192
##     obama       142   656
```

The correct ratio of goodness of fit is as follows:

```
sum(diag(table(predict(model2_knn_n,scaled_train), scaled_train$winner))/(sum(rowSums(table(predict(mode
```

```
## [1] 0.8276749
```

It seems like the initial KNN model yields better classifications than the current optimal `k` = 21 model for the train dataset. Let's see if there is indded difference in classification of prediction for the test dataset.

The table below presents the correct classifications for the initial model:

```r
table(predictions = predict(model1_knn5,scaled_test), obsereved =  scaled_test$winner)
```

```
##            obsereved
## predictions clinton obama
##     clinton     217    44
##     obama        56   168
```

Below we get to rate of correct classification for the initial model:

```r
sum(diag(table(predict(model1_knn5,scaled_test), scaled_test$winner))/(sum(rowSums(table(predict(model1_
```

```
## [1] 0.7938144
```

The table below presents the correct classifications of the second model with `k` = 21:

```r
table(predictions = predict(model2_knn_n,scaled_test), obsereved =  scaled_test$winner)
```

```
##            obsereved
## predictions clinton obama
##     clinton     231    46
##     obama        42   166
```

Below we get to rate of correct classification for the second model:

```r
sum(diag(table(predict(model2_knn_n,scaled_test), scaled_test$winner))/(sum(rowSums(table(predict(model2
```

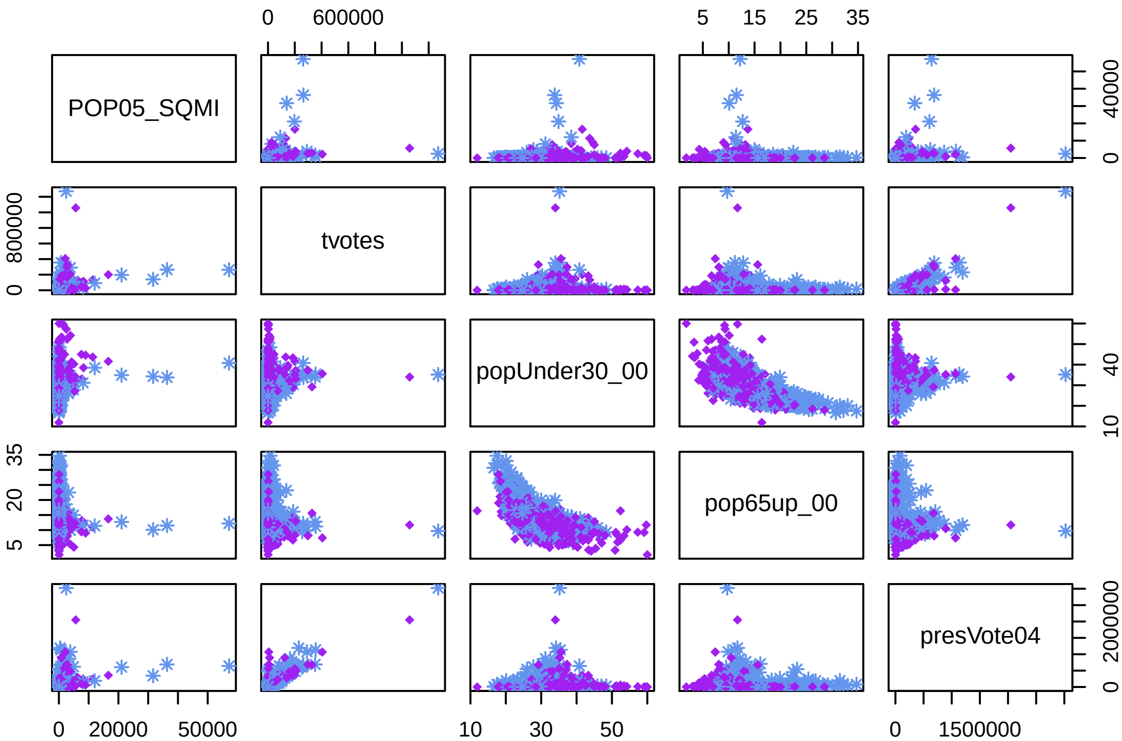
```
## [1] 0.8185567
```

Indeed it is clear to see that the optimized model with `k` = 21 has better performance on the test dataset and that it should be the best model in this KNN section.

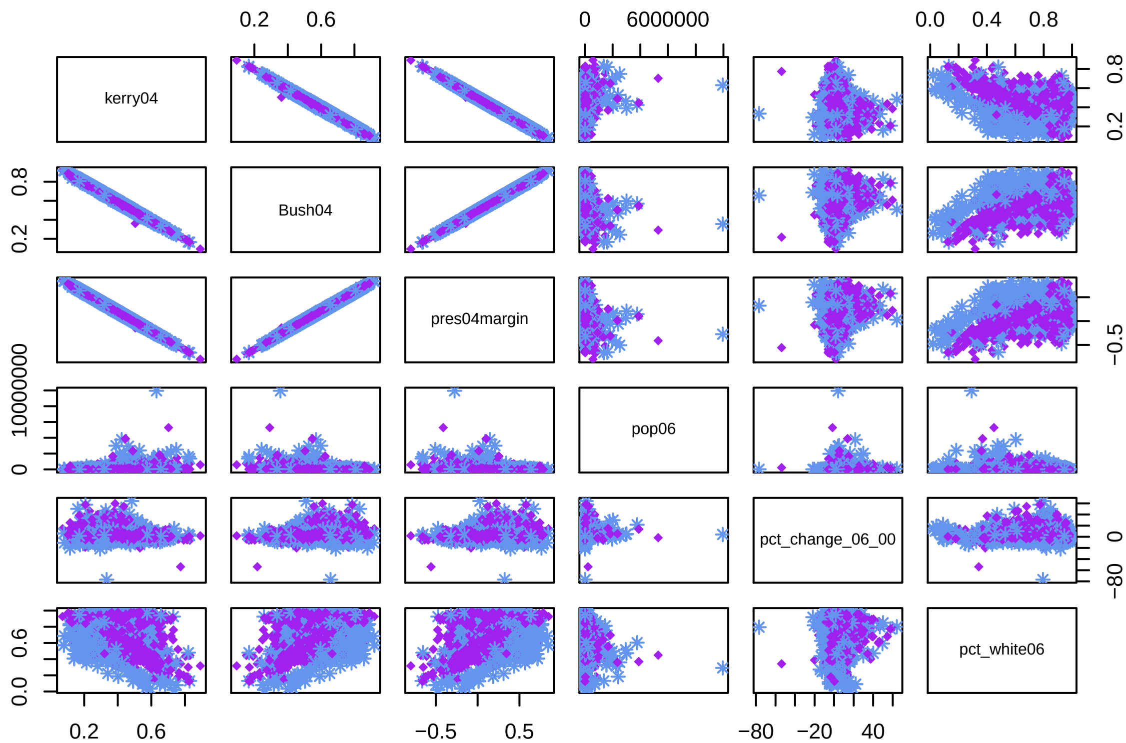
SVM: Support Vector Machine

The last modeling is the SVM. First step in the SVM modeling is to have a look at different pairs of variables the the way their are distributed but more importantly is how the dependent variable, the `winner`, is spread in the distribution. The plots of the distributions is as follows:

```r
pairs(primaries[,c(1:6)] %>% keep(is.numeric),
      col = c("cornflowerblue", "purple")[primaries$winner],   # Change color by group
      pch = c(8, 18)[primaries$winner])
```



POP05_SQMI
tvotes
popUnder30_00
pop65up_00
presVote04
0
600000
5
15
25
35
40000
0
800000
0
40
10
35
20
5
2000000
0
0
20000
50000
10
30
50
0
1500000



As seen above it looks very clearly that a linear separation would not be sufficient to make good predictions. Moreover, it seems like for most variables, polynomial seperation would not be fit either.
The best method would be Gaussian Radial Kernel seperation, as most of the distribution, one level of `winner` is "swallowed" and surrounded by the other.

The first model would be be a "basic" radial SVM model with all the variables as predictors, no cross validation, and all parameters are set to default:

```
set.seed(123321)
model1.svm <- train(winner ~.,
                            data = primaries_train,
                            method = "svmRadial")
```

More details on the defaut model is below:

```
model1.svm$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.025352925106861
##
## Number of Support Vectors : 1054
##
```

```
## Objective Function Value : -787.14
## Training error : 0.119342
```

We can see that the `Cost` parameter was automatically set to be 1 and `sigma` to be 0.02535. Let's take a look at the confusion matrix output below:

```
confusionMatrix(
  predict(model1.svm, primaries_train),
   primaries_train$winner,
   positive = "obama"
)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction clinton obama
##    clinton     992   142
##    obama       104   706
##
##                Accuracy : 0.8735
##                  95% CI : (0.8579, 0.8879)
##     No Information Rate : 0.5638
##     P-Value [Acc > NIR] : < 0.0000000000000002
##
##                   Kappa : 0.7414
##
##  Mcnemar's Test P-Value : 0.01832
##
##             Sensitivity : 0.8325
##             Specificity : 0.9051
##          Pos Pred Value : 0.8716
##          Neg Pred Value : 0.8748
##              Prevalence : 0.4362
##          Detection Rate : 0.3632
##    Detection Prevalence : 0.4167
##       Balanced Accuracy : 0.8688
##
##        'Positive' Class : obama
##
```

Even a simple basic Radial SVM model we preform nice accuracy rate of 0.8735 in classifying the `winner`. We will try to improve that basic model with cross validation. We start with taking different Cost level (`C`) and `sigma`'s as follows:

```
df_cSigma <-
  expand.grid(C = c(0.01, 0.05, 0.1, 0.5, 1,3, 5),
              sigma = c(seq(0.01,0.05,0.01), 0.08, 0.1, 0.2, 0.5, 1))

headtail(df_cSigma)
```

```
##        C sigma
## 1   0.01  0.01
## 2   0.05  0.01
```

```
## 3    0.1  0.01
## 4    0.5  0.01
## ...  ...   ...
## 67   0.5     1
## 68     1     1
## 69     3     1
## 70     5     1
```

We use repeated cross validation with 5 folds which repeated 3 times (15 times in total) in order to get the best model with best `C` and `sigma` (from the data frame above) which will maximize our model results.

```
myTrainCont <- trainControl(method = "repeatedcv",
                            number = 5,
                            repeats = 3)
```

```
model2.svm <- train(winner ~.,
                            data = primaries_train,
                            tuneGrid = df_cSigma,     #two cells above
                            trControl = myTrainCont, #one cell above
                            method = "svmRadial")
```

Below are the details for the best model which was optimized by the cross validation process:

```
model2.svm$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 3
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.03
##
## Number of Support Vectors : 953
##
## Objective Function Value : -1731.697
## Training error : 0.077675
```

The optimal cost and sigma are 3 and 0.03 respectivly.

Below is the confusion matrix for the current model perdictions:

```
confusionMatrix(
  predict(model2.svm, primaries_train),
   primaries_train$winner,
   positive = "obama"
)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction clinton obama
```

```
##    clinton    1019    90
##    obama        77   758
##
##                Accuracy : 0.9141
##                  95% CI : (0.9007, 0.9262)
##     No Information Rate : 0.5638
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.825
##
##  Mcnemar's Test P-Value : 0.3531
##
##             Sensitivity : 0.8939
##             Specificity : 0.9297
##          Pos Pred Value : 0.9078
##          Neg Pred Value : 0.9188
##              Prevalence : 0.4362
##          Detection Rate : 0.3899
##    Detection Prevalence : 0.4295
##       Balanced Accuracy : 0.9118
##
##        'Positive' Class : obama
##
```

We get quite surprising results as the accuracy of the optimized SVM model is : 0.9141!
Let's check if the model preforms as good on the test dataset as it did on the train dataset:

```r
confusionMatrix(
  predict(model2.svm, primaries_test),
   primaries_test$winner,
   positive = "obama"
)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction clinton obama
##    clinton     230    20
##    obama        43   192
##
##                Accuracy : 0.8701
##                  95% CI : (0.8369, 0.8987)
##     No Information Rate : 0.5629
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.7392
##
##  Mcnemar's Test P-Value : 0.005576
##
##             Sensitivity : 0.9057
##             Specificity : 0.8425
##          Pos Pred Value : 0.8170
##          Neg Pred Value : 0.9200
```

```
##               Prevalence : 0.4371
##           Detection Rate : 0.3959
##     Detection Prevalence : 0.4845
##        Balanced Accuracy : 0.8741
##
##         'Positive' Class : obama
##
```

We can see that model still preforms well on the test dataset. On one hand, the model doesn't predict as good as it did on the train dataset, but on the other hand it looks like it performs better on tests dataset than the other models! We shall check that on the next section of model comparison.

Models Comparison and Conclusion

Akaike Information Criterion (AIC) is a criterion score which rewards for goodness of fit and panelize for complexity. We use AIC to compare models to determine which one is the best from all the alternatives. There are 2 conditions before we can compare models with AIC: the dependent variable should be the same for all models, as well as the dataset. The former condition is met in all 3 models, however the dataset is different in KNN, as the dataset was normalized.
Let us compare the goodness of fit for all selected models, `model3` of logit, `model2_knn_n` off KNN and `model2.svm` of SVM, for the test dataset:

For the logit best model, `model3`, the accuracy is 80.4% correct classifications:

```
confusionMatrix(
factor(ifelse(predict(model3, newdata = primaries_test, type = "response")>=0.47,"obama","clinton") ),
factor(primaries_test$winner),
positive = "obama"
)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction clinton obama
##    clinton     218    39
##    obama        55   173
##
##                Accuracy : 0.8062
##                  95% CI : (0.7681, 0.8404)
##     No Information Rate : 0.5629
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.6094
##
##  Mcnemar's Test P-Value : 0.1218
##
##             Sensitivity : 0.8160
##             Specificity : 0.7985
##          Pos Pred Value : 0.7588
##          Neg Pred Value : 0.8482
##              Prevalence : 0.4371
##          Detection Rate : 0.3567
##    Detection Prevalence : 0.4701
```

```
##       Balanced Accuracy : 0.8073
##
##        'Positive' Class : obama
##
```

For the KNN best model, `model2_knn_n`, the accuracy is 82.8% correct classifications:

```
confusionMatrix(
predict(model2_knn_n, newdata = scaled_test),
primaries_test$winner)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction clinton obama
##    clinton     231    46
##    obama        42   166
##
##                Accuracy : 0.8186
##                  95% CI : (0.7813, 0.8519)
##     No Information Rate : 0.5629
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.6305
##
##  Mcnemar's Test P-Value : 0.7491
##
##             Sensitivity : 0.8462
##             Specificity : 0.7830
##          Pos Pred Value : 0.8339
##          Neg Pred Value : 0.7981
##              Prevalence : 0.5629
##          Detection Rate : 0.4763
##    Detection Prevalence : 0.5711
##       Balanced Accuracy : 0.8146
##
##        'Positive' Class : clinton
##
```

And for the KNN best model, `model2.svm`, the accuracy is 87% correct classifications:

```
confusionMatrix(
  predict(model2.svm, primaries_test),
   primaries_test$winner,
   positive = "obama"
)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction clinton obama
##    clinton     230    20
```

```
##    obama         43   192
##
##                Accuracy : 0.8701
##                  95% CI : (0.8369, 0.8987)
##     No Information Rate : 0.5629
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.7392
##
##  Mcnemar's Test P-Value : 0.005576
##
##             Sensitivity : 0.9057
##             Specificity : 0.8425
##          Pos Pred Value : 0.8170
##          Neg Pred Value : 0.9200
##              Prevalence : 0.4371
##          Detection Rate : 0.3959
##    Detection Prevalence : 0.4845
##       Balanced Accuracy : 0.8741
##
##        'Positive' Class : obama
##
```

From the outputs above it's obvious that SVM performs the best on both train and test datasets in all parameters: Balanced Accuracy, Sensitivity and Specificity. However there is a downside to it, it dements lots of processing resourses and time. The former models were preforming well relatively to the time and computational resources.
For big datasets SVM is mostly not suited if the resources are limited and one might as well switch to other methods such as Logistic, Probit, KNN and other classification algorithms.
For our case SVM seems to be the best option!

Thank you for reading!