**CS 6071: Midterm Examination**

**Date: October 27th 2015**

**UNIVERSITY OF CINCINNATI HONOR PLEDGE**

On my honor I pledge that this work of mine does not violate the University of Cincinnati Student Code of Conduct provisions on cheating and plagiarism.

**Printed name:** Charu Khatwani

**Signature:** Charu Khatwani

**Initial Data Exploration and Understanding**

When started to work on FEC data my first aim was to understand the significance and relevance of each and every feature and finding out the most suitable amongst the in terms of feature importance.

**Step 0 – Header files**

\_\_author\_\_ = **'charukhatwani'***#Array processing***import** subprocess  
**import** numpy **as** np  
  
**from** sklearn **import** datasets  
**from** sklearn **import** cross\_validation  
**from** sklearn **import** preprocessing  
**from** sklearn.metrics **import** confusion\_matrix,accuracy\_score,classification\_report  
**from** sklearn.cross\_validation **import** StratifiedKFold  
*#Data analysis, wrangling and common exploratory operations***import** pandas **as** pd  
**from** pandas **import** Series, DataFrame  
  
*#For visualization. Matplotlib for basic viz and seaborn for more stylish figures + statistical figures not in MPL.***import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sns  
  
*# Decision tree classifier***from** sklearn.tree **import** DecisionTreeClassifier, export\_graphviz  
**from** sklearn.ensemble **import** RandomForestClassifier  
  
**import** pandas **as** pd

**Step 1- Loading the data**

fec = pd.read\_csv(**'P00000001-ALL-2008.csv'**,index\_col=False)

**print** fec.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 4085665 entries, 0 to 4085664

Data columns (total 18 columns):

cmte\_id object

cand\_id object

cand\_nm object

contbr\_nm object

contbr\_city object

contbr\_st object

contbr\_zip object

contbr\_employer object

contbr\_occupation object

contb\_receipt\_amt float64

contb\_receipt\_dt object

receipt\_desc object

memo\_cd object

memo\_text object

form\_tp object

file\_num int64

tran\_id object

election\_tp object

dtypes: float64(1), int64(1), object(16)

memory usage: 592.3+ MB

None

**Sample Record**

cmte\_id C00430470

cand\_id P80002801

cand\_nm McCain, John S

contbr\_nm EATON, ROBERT J. MR.

contbr\_city NAPLES

contbr\_st FL

contbr\_zip 341081997

contbr\_employer RETIRED

contbr\_occupation RETIRED

contb\_receipt\_amt 2300

contb\_receipt\_dt 26-FEB-08

receipt\_desc NaN

memo\_cd NaN

memo\_text NaN

form\_tp SA17A

file\_num 876806

tran\_id SA17.704365

election\_tp P2008

**Step 2- Understanding the data – Data knowledge**

1. Finding out the candidates for 2008,2012 and 2016 presidential elections and developing a dictionary for each candidates to Map to their respective parties

Code -  
unique\_cands\_2008= fec.cand\_nm.unique()  
unique\_cands\_2012= fec\_2012.cand\_nm.unique()  
unique\_cands\_2016= fec\_2016.cand\_nm.unique()  
  
**print '2008 candidates'  
print** unique\_cands\_2008  
**print '2012 candidates'  
print** unique\_cands\_2012  
**print '2016 candidates'  
print** unique\_cands\_2016

Output-

**2008 candidates**

['Cox, John H' 'Gravel, Mike' 'McCain, John S' 'Giuliani, Rudolph W' 'Brownback, Samuel Dale' 'Thompson, Tommy G' 'Kucinich, Dennis J’ 'Romney, Mitt' 'Edwards, John' 'Gilmore, James S III''Dodd, Christopher J' 'Hunter, Duncan' 'Obama, Barack''Clinton, Hillary Rodham' 'Richardson, Bill' 'Tancredo, Thomas Gerald''Huckabee, Mike' 'Biden, Joseph R Jr' 'Paul, Ron' 'Thompson, Fred Dalton']

**2012 candidates**

['Bachmann, Michele' 'Romney, Mitt' 'Obama, Barack'"Roemer, Charles E. 'Buddy' III" 'Pawlenty, Timothy' 'Johnson, Gary Earl’ 'Paul, Ron' 'Santorum, Rick' 'Cain, Herman' 'Gingrich, Newt' 'McCotter, Thaddeus G' 'Huntsman, Jon' 'Perry, Rick' 'Stein, Jill']

**2016 candidates**

['Rubio, Marco' 'Santorum, Richard J.' 'Perry, James R. (Rick)''Carson, Benjamin S.' "Cruz, Rafael Edward 'Ted'" 'Paul, Rand''Clinton, Hillary Rodham' 'Sanders, Bernard' 'Fiorina, Carly''Huckabee, Mike' 'Pataki, George E.' "O'Malley, Martin Joseph" 'Graham, Lindsey O.' 'Bush, Jeb' 'Trump, Donald J.' 'Jindal, Bobby'

'Christie, Christopher J.' 'Walker, Scott' 'Webb, James Henry Jr.''Kasich, John R.' 'Lessig, Lawrence']

1. Linking each candidate to their respective parties

*#while doing analysis later we want to associate each candidate with their party*parties = {**'Bachmann, Michele'**: **'Republican'**,  
 **'Cain, Herman'**: **'Republican'**,  
 **'Gingrich, Newt'**: **'Republican'**,  
 **'Huntsman, Jon'**: **'Republican'**,  
 **'Johnson, Gary Earl'**: **'Republican'**,  
 **'McCotter, Thaddeus G'**: **'Republican'**,  
 **'Obama, Barack'**: **'Democratic'**,  
 **'Paul, Ron'**: **'Republican'**,  
 **'Pawlenty, Timothy'**: **'Republican'**,  
 **'Perry, Rick'**: **'Republican'**,  
 **"Roemer, Charles E. 'Buddy' III"**: **'Republican'**,  
 **'Romney, Mitt'**: **'Republican'**,  
 **'Santorum, Rick'**: **'Republican'**,  
 **'Stein, Jill'**:**'Green'**,  
 **'Rubio, Marco'**:**'Republican'**,  
 **'Santorum, Richard J.'**:**'Republican'**,  
 **'Perry, James R. (Rick)'**:**'Republican'**,  
 **'Carson, Benjamin S.'**:**'Republican'**,  
 **"Cruz, Rafael Edward 'Ted'"**:**'Republican'**,  
 **'Paul, Rand'**:**'Republican'**,  
 **'Clinton, Hillary Rodham'**:**'Democratic'**,  
 **'Sanders, Bernard'**:**'Democratic'**,  
 **'Fiorina, Carly'**:**'Republican'**,  
 **'Huckabee, Mike'**:**'Republican'**,  
 **'Pataki, George E.'**:**'Republican'**,  
 **"O'Malley, Martin Joseph"**:**'Democratic'**,  
 **'Graham, Lindsey O.'**:**'Republican'**,  
 **'Bush, Jeb'**:**'Republican'**,  
 **'Trump, Donald J.'**:**'Republican'**,  
 **'Jindal, Bobby'**:**'Republican'**,  
 **'Christie, Christopher J.'**:**'Republican'**,  
 **'Walker, Scott'**:**'Republican'**,  
 **'Webb, James Henry Jr.'**:**'Republican'**,  
 **'Kasich, John R.'**:**'Republican'**,  
 **'Lessig, Lawrence'**:**'Democratic'**,  
 **'Huckabee, Mike'**: **'Republican'**,  
 **'Paul, Ron'**: **'Republican'**,  
 **'Hunter, Duncan'**: **'Republican'**,  
 **'Thompson, Fred Dalton'**: **'Republican'**,  
 **'Richardson, Bill'**: **'Democratic'**,  
 **'McCain, John S'**: **'Republican'**,  
 **'Clinton, Hillary Rodham'**: **'Democratic'**,  
 **'Edwards, John'**: **'Democratic'**,  
 **'Giuliani, Rudolph W'**: **'Republican'**,  
 **'Brownback, Samuel Dale'**:**'Republican'**,  
 **'Tancredo, Thomas Gerald'**: **'Republican'**,  
 **'Cox, John H'**:**'Republican'**,  
 **'Biden, Joseph R Jr'**: **'Democratic'**,  
 **'Gravel, Mike'**: **'Democratic'**,  
 **'Dodd, Christopher J'**:**'Democratic'**,  
 **'Kucinich, Dennis J'**:**'Democratic'**,  
 **'Gilmore, James S III'**:**'Republican'**,  
 **''**:**'Other'**}

1. Adding it as a column in data frame

fec[**'party'**] = fec.cand\_nm.map(parties)  
fec\_2012[**'party'**] = fec\_2012.cand\_nm.map(parties)  
fec\_2016[**'party'**] = fec\_2016.cand\_nm.map(parties)

1. Calculating year wise party votes for 2008,2012 and 2016 respectively-

*#Calculating the counts of each year party wise***print '2008 party wise vote counts'  
print** fec[**'party'**].value\_counts()  
  
**print '2012 party wise vote counts'  
print** fec\_2012[**'party'**].value\_counts()  
  
**print '2016 party wise vote counts'  
print** fec\_2016[**'party'**].value\_counts()

**Output-**

**2008 party wise vote counts**

Democratic 3293707

Republican 791064

Name: party, dtype: int64

**2012 party wise vote counts**

Democratic 4117404

Republican 1917737

Green 1317

Name: party, dtype : int64

**2016 party wise vote counts**

Republican 227328

Democratic 157557

Name: party, dtype: int64

**Data Cleaning and Assumptions**

One main data cleaning operation I did was trimming the zip code and fetching the first 5 digits as USA has only 5digit zip code.

Also, there were some invalid zip codes in OHIO state for 2012 like ‘4421s,which I have treated as 44212,considering the city wise main zip code.

Similarly,other invalid zip codes were present and invalid occupation names which I had to clean in order to go forward with the model.

**Model Selection-**

I have chosen Random forest as the predictor as it creates a bunch of decision trees and one out of all is highly accurate. It is better than SVM and KNN and Decision Tree Classifier in terms of accuracy.

**Description-**

It belongs to a larger class of machine learning algorithms called ensemble methods.

Ensemble Learning involves the combination of several models to solve a single prediction problem. It works by generating multiple classifiers/models which learn and make predictions independently. Those predictions are then combined into a single (mega) prediction that should be as good or better than the prediction made by any one classifier.

So we know that random forest is an aggregation of other models, but what types of models is it aggregating? Random forest aggregates [Classification (or Regression) Trees](http://en.wikipedia.org/wiki/Decision_tree_learning). A decision tree is composed of a series of decisions that can be used to classify an observation in a dataset.

**Answers**

1. Using machine learning build a set of models for each state that predict the winner of the popular vote. Detail the most important features for each of these models. Document your methods for validation.

The model uses 3 main features to predict the win for each state.

**The features are-**

* 1. Contribution Occupation – Very strong predictor as explained above.
  2. Contribution amount
  3. Zip code

**contb\_receipt\_amt'**,**'zip\_valid'**,**'contbr\_occupation\_id**

**Assumptions –**

1. Contribution amount >0 so as to capture only the positive contribution amount
2. Zip code only first 5 digits
3. Each occupation is different e.g INFORMATION REQUESTED PER BEST EFFORTS', 'INFORMATION REQUESTED'  are different occupations.

Below I will depict the model for OHIO state and its method of validation in steps-

**Results and Validation Output for OHIO State-**

**Scores**

[ 0.87893616 0.91025897 0.88958791 0.88622216 0.89883489]

**Accuracy**: 0.89 (+/- 0.02)

**Validation :**

[ 41392 41393 41394 ..., 150204 150205 150206] [ 0 1 2 ..., 56215 56216 56217]

[ 0 1 2 ..., 150204 150205 150206] [ 41392 41393 41394 ..., 105353 105354 105355]

[ 0 1 2 ..., 105353 105354 105355] [ 41685 42093 42094 ..., 150204 150205 150206]

Validation –skfold used

**Feature Importance**

Clf.feature\_importances\_

It depicts that Contribution amount is the most important feature for prediction.

**Results**

2016 results

Democrat Wins in 2016 as per the Predictive Model for state OH

Confusion matrix ,without normalization

[[61083 0 29505]

[ 13 0 6]

[31513 0 28087]]

**precision recall f1-score support**

**class 0 0.66 0.67 0.67 90588**

**class 1 0.00 0.00 0.00 19**

**class 2 0.49 0.47 0.48 59600**

**avg / total 0.59 0.59 0.59 150207**

****

STEP 1- ZIP code trimming and cleaning of invalid zip codes from data

*#We tend to use zip code as a categorial variable and trimming the zip value to contain only 5 characters*

*-------2008 data*fec[**"zip\_valid"**]=(fec.contbr\_zip).map(**lambda** x:str(x)[:5])  
fec[**"zip\_valid"**]=(fec.contbr\_zip).map({**'NG7 1'**: 12345,**'V6R 3M9'**: 12345,**'M5T 2'**: 12345,**'4221s'**:12345})  
**print** fec.zip\_valid[fec.zip\_valid.isin([**'M5T 2'**])]  
  
fec[**"zip\_valid"**]=pd.DataFrame(fec[**"zip\_valid"**].astype(float))  
fec[**"zip\_valid"**]=fec[**"zip\_valid"**].fillna(value=0)  
  
*-------2012 data*  
fec\_2012[**"zip\_valid"**]=(fec\_2012.contbr\_zip).map(**lambda** x:str(x)[:5])  
fec\_2012[**"zip\_valid"**]=(fec\_2012.contbr\_zip).map({**'NG7 1'**: 12345,**'V6R 3M9'**: 12345,**'M5T 2'**: 12345,**'4221s'**:12345})  
  
fec\_2012[**"zip\_valid"**]=pd.DataFrame(fec\_2012[**"zip\_valid"**].astype(float))  
fec\_2012[**"zip\_valid"**]=fec\_2012[**"zip\_valid"**].fillna(value=0)

*-------2016 data*  
fec\_2016[**"zip\_valid"**]=(fec\_2016.contbr\_zip).map(**lambda** x:str(x)[:5])  
fec\_2016[**"zip\_valid"**]=(fec\_2016.contbr\_zip).map({**'NG7 1'**: 12345,**'V6R 3M9'**: 12345,**'M5T 2'**: 12345,**'4221s'**:12345})  
  
fec\_2016[**"zip\_valid"**]=pd.DataFrame(fec\_2016[**"zip\_valid"**].astype(float))  
fec\_2016[**"zip\_valid"**]=fec\_2016[**"zip\_valid"**].fillna(value=0)

**STEP 2 – Mapping Occupation with respective Occupation ID’s to use it as a feature**

*#Random forest treat int values as categorial  
  
  
#using occupation as a feature*temp\_fec = pd.DataFrame({**'contbr\_occupation'**: fec.contbr\_occupation.unique(), **'contbr\_occupation\_id'**:range(len(fec.contbr\_occupation.unique()))})  
fec = fec.merge(temp\_fec, on=**'contbr\_occupation'**, how=**'left'**)  
  
  
*# Adding occupation id column for occupation names 2012*temp\_fec\_2012 = pd.DataFrame({**'contbr\_occupation'**: fec\_2012.contbr\_occupation.unique(), **'contbr\_occupation\_id'**:range(len(fec\_2012.contbr\_occupation.unique()))})  
fec\_2012 = fec\_2012.merge(temp\_fec\_2012, on=**'contbr\_occupation'**, how=**'left'**)  
  
  
  
*# Adding occupation id column for occupation names 2016*temp\_fec\_2016 = pd.DataFrame({**'contbr\_occupation'**: fec\_2016.contbr\_occupation.unique(), **'contbr\_occupation\_id'**:range(len(fec\_2016.contbr\_occupation.unique()))})  
fec\_2016 = fec\_2016.merge(temp\_fec\_2016, on=**'contbr\_occupation'**, how=**'left'**)

**STEP 3- Developing the random forest model using the features**

TrainingFeatures = fec[[**'contb\_receipt\_amt'**,**'zip\_valid'**,**'contbr\_occupation\_id'**]]  
TrainingClassLabels = fec[**'party'**]  
classifier = RandomForestClassifier(n\_estimators=100)  
clf = classifier.fit(TrainingFeatures,TrainingClassLabels)

**STEP 4- Testing on 2012 data**

*#Testing on 2012 Data*TestFeatures = fec\_2012[[**'contb\_receipt\_amt'**,**'zip\_valid'**,**'contbr\_occupation\_id'**]]  
TestClassLabels=fec\_2012[**'party'**]  
Test\_predict=clf.predict(TestFeatures)

**STEP 5- Cross validating the 2012 model**

*#Cross Validating on 2012 data*ValidationFeatures = fec\_2012[[**'contb\_receipt\_amt'**,**'zip\_valid'**,**'contbr\_occupation\_id'**]]  
ValidationClassLabels=fec\_2012[**'party'**]  
  
scores=cross\_validation.cross\_val\_score(clf,ValidationFeatures,ValidationClassLabels,cv=5)

**print 'Scores'  
print** scores

**STEP 6 – Accuracy calculation**

**print**(**"Accuracy: %0.2f (+/- %0.2f)"** % (scores.mean(), scores.std() \* 2))  
  
*#K fold cross validation*labels = fec\_2012.party  
  
skf=StratifiedKFold(labels,3)  
**for** train,test **in** skf:  
 **print**(**"%s %s"**%(train,test))  
  
**STEP 7 – Predicting 2016 elections**

*#Now we will predict 2016 results*PredictFeatures = fec\_2016[[**'contb\_receipt\_amt'**,**'zip\_valid'**,**'contbr\_occupation\_id'**]]  
*#TestClassLabels=fec\_2012['party']*Predict2016 = clf.predict(PredictFeatures)  
  
**print** len(Predict2016)  
fec\_2016[**"predict\_winner"**] = Predict2016  
  
*#For each state, predict winner of popular vote.***print '2016 results'**state = fec\_2016.contbr\_st.unique()  
**for** j **in** range(len(state)):  
 **for** i **in** range(len(fec\_2016[**"predict\_winner"**])) :  
 **if** fec\_2016.contbr\_st[i]==state[j]:  
 Democrat=0  
 Republican=0  
 Other=0  
 **if** fec\_2016.predict\_winner[i]==**'Democratic'**: *#and fec\_2016.contbr\_st[i]==state[j]:* Democrat=Democrat+1  
 **elif** fec\_2016.predict\_winner[i]==**'Republican'**: *#and fec\_2016.contbr\_st[i]==state[j]:* Republican=Republican+1  
 **else**:  
 Other=Other+1  
 **if** Democrat>Republican:  
  
 **print 'Democrat Wins in 2016 as per the Predictive Model for state'  
 print** state[j]  
 **else**:  
 **print 'Republican Wins in 2016 as per the Predictive Model for state'  
 print** state[j]  
  
  
  
  
  
  
  
cm=confusion\_matrix(TestClassLabels,Test\_predict)  
*#print cm*np.set\_printoptions(precision=2)  
**print**(**'Confusion matrix ,without normalization'**)  
**print**(cm)  
**print 'Accuracy'  
print** accuracy\_score(TestClassLabels, Test\_predict, normalize=False)  
  
labels=[**'Democratic'**,**'Republicans'**]  
*#plt.matshow(cm)  
#plt.title('Normalized Confusion matrix')  
#plt.colorbar()*fig = plt.figure()  
ax = fig.add\_subplot(111)  
cax = ax.matshow(cm)  
plt.title(**''Confusion matrix ,without normalization**)  
fig.colorbar(cax)  
ax.set\_xticklabels([**''**] + labels)  
ax.set\_yticklabels([**''**] + labels)  
plt.ylabel(**'True label'**)  
plt.xlabel(**'Predicted label'**)  
plt.show()  
  
  
*#print clf.feature\_importances\_*cm\_normalized=cm.astype(**'float'**)/cm.sum(axis=1)[:,np.newaxis]  
**print**(**'Normalized Confusion Matrix'**)  
**print**(cm\_normalized)  
**print 'Normalized Accuracy'  
print** accuracy\_score(TestClassLabels, Test\_predict)  
labels=[**'Democratic'**,**'Republicans'**]  
*#plt.matshow(cm)  
#plt.title('Normalized Confusion matrix')  
#plt.colorbar()*fig = plt.figure()  
ax = fig.add\_subplot(111)  
cax = ax.matshow(cm\_normalized)  
plt.title(**'Normalized Confusion matrix of the classifier'**)  
fig.colorbar(cax)  
ax.set\_xticklabels([**''**] + labels)  
ax.set\_yticklabels([**''**] + labels)  
plt.ylabel(**'True label'**)  
plt.xlabel(**'Predicted label'**)  
plt.show()  
target\_names = [**'class 0'**, **'class 1'**,**'class 2'**]  
  
**print**(classification\_report(TestClassLabels, Test\_predict, target\_names=target\_names))

1. Using machine learning build a model for the national election that predicts the winner of the popular vote. Detail the most important features for your models. Document your methods for validation.

Model-

**Democrat wins in 45 states .**

**Republican wins in 35 states**

**So according to my Model the Democrat wins the national election as it dominates the electoral votes of 45 states.**

**print '2016 results'**Final\_Democrat=0  
Final\_Republican=0  
state = fec\_2016.contbr\_st.unique()  
**for** j **in** range(len(state)):  
 **for** i **in** range(len(fec\_2016[**"predict\_winner"**])) :  
 **if** fec\_2016.contbr\_st[i]==state[j]:  
 Democrat=0  
 Republican=0  
 Other=0  
 **if** fec\_2016.predict\_winner[i]==**'Democratic'**: *#and fec\_2016.contbr\_st[i]==state[j]:* Democrat=Democrat+1  
 **elif** fec\_2016.predict\_winner[i]==**'Republican'**: *#and fec\_2016.contbr\_st[i]==state[j]:* Republican=Republican+1  
 **else**:  
 Other=Other+1  
 **if** Democrat>Republican:  
  
 **print 'Democrat Wins in 2016 as per the Predictive Model for state'  
 print** state[j]  
 Final\_Democrat=Final\_Democrat+1  
 **else**:  
 **print 'Republican Wins in 2016 as per the Predictive Model for state'  
 print** state[j]  
 Final\_Republican=Final\_Republican+1

The model built runs on each and every state and then predicts the win on the on following features state wise-

1.Contribution Occupation – Very strong predictor as explained above.

2.Contribution amount

3.Zip code

TrainingFeatures = fec[[**'contb\_receipt\_amt'**,**'zip\_valid'**,**'contbr\_occupation\_id'**]]  
TrainingClassLabels = fec[**'party'**]  
classifier = RandomForestClassifier(n\_estimators=100)  
clf = classifier.fit(TrainingFeatures,TrainingClassLabels)

Classifier used – Random forest with accuracy 0.87

**Methods of validation**

Accuracy: 0.81 (+/- 0.03)

I have used the cross validation [**cross\_val\_score**](http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.cross_val_score.html#sklearn.cross_validation.cross_val_score) metric to cross validate the model on OHIO state.

ValidationFeatures = fec\_2012[[**'contb\_receipt\_amt'**,**'zip\_valid'**,**'contbr\_occupation\_id'**]]  
ValidationClassLabels=fec\_2012[**'party'**]  
  
scores=cross\_validation.cross\_val\_score(clf,ValidationFeatures,ValidationClassLabels,cv=5)  
**print 'Scores'  
print** scores  
**print**(**"Accuracy: %0.2f (+/- %0.2f)"** % (scores.mean(), scores.std() \* 2))

1. **Using machine learning build a model for the likelihood of an individual to contribute to Democrats, or Republicans. Detail the most important features for each of these models. Document your methods for validation.**

The most important features here are Contributor occupation and the contribution amount.Below is a step by step explanation of using those features.

Likelihood – If a person is retired/attorney/homemaker/physician he/she will contribute the most as depicted below.

RETIRED 1388962

ATTORNEY 195799

HOMEMAKER 154703

PHYSICIAN 141315

**Methods of validation**

Accuracy: 0.81 (+/- 0.03)

I have used the cross validation [**cross\_val\_score**](http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.cross_val_score.html#sklearn.cross_validation.cross_val_score) metric to cross validate the model on OHIO state.

ValidationFeatures = fec\_2012[[**'contb\_receipt\_amt'**,**'zip\_valid'**,**'contbr\_occupation\_id'**]]  
ValidationClassLabels=fec\_2012[**'party'**]  
  
scores=cross\_validation.cross\_val\_score(clf,ValidationFeatures,ValidationClassLabels,cv=5)  
**print 'Scores'  
print** scores  
**print**(**"Accuracy: %0.2f (+/- %0.2f)"** % (scores.mean(), scores.std() \* 2))

1. To simplify the model analysis, I will restrict only to positive contributions

fec = fec[fec.contb\_receipt\_amt > 0]   
fec\_2012[**'party'**] = fec\_2012.cand\_nm.map(parties)  
fec\_2016[**'party'**] = fec\_2016.cand\_nm.map(parties)

1. Donation by occupation is yet another feature that is considerable -Lawyers and Doctors tend to donate more than other people. Hence it can be a very good Feature

*#Top 10 Contribution Statistics by Occupation*Contribution\_2008=fec.contbr\_occupation.value\_counts()[:10]  
**print '2008 top 10 contributions by occupation'   
print** Contribution\_2008

**OUTPUT**

2008 top 10 contributions by occupation

RETIRED 768150

ATTORNEY 224249

NOT EMPLOYED 119542

HOMEMAKER 86813

PHYSICIAN 86723

INFORMATION REQUESTED 86347

PROFESSOR 71495

CONSULTANT 66901

TEACHER 64316

ENGINEER 40409

Name: contbr\_occupation, dtype: int64

Contribution\_2012=fec\_2012.contbr\_occupation.value\_counts()[:10]  
**print '2012 top 10 contributions by occupation'   
print** Contribution\_2012

**2012 top 10 contributions by occupation**

RETIRED 1388962

ATTORNEY 195799

INFORMATION REQUESTED PER BEST EFFORTS 173593

HOMEMAKER 154703

PHYSICIAN 141315

INFORMATION REQUESTED 125136

TEACHER 105559

PROFESSOR 96457

CONSULTANT 80411

ENGINEER 75792

Name: contbr\_occupation, dtype: int64

Contribution\_2016=fec\_2016.contbr\_occupation.value\_counts()[:10]  
**print '2016 top 10 contributions by occupation'   
print** Contribution\_2016

2016 top 10 contributions by occupation

RETIRED 96007

NOT EMPLOYED 22639

INFORMATION REQUESTED PER BEST EFFORTS 14296

ATTORNEY 13174

HOMEMAKER 10238

PHYSICIAN 7863

INFORMATION REQUESTED 7111

CONSULTANT 5168

ENGINEER 4414

CEO 3940

Name: contbr\_occupation, dtype: int64

So, we can see the trends for 2008,2012 and 2016,where the Retired people tend to donate the most, where as Attorney , Homemaker and Doctors occupy the subsequent positions.

1. ***Now we see the bifurcation donation party wise***

by\_occupation = fec.pivot(**'contb\_receipt\_amt'**, rows=**'contbr\_occupation'**,  
cols=**'party'**, aggfunc=**'sum'**)  
  
over\_1mm\_2008 = by\_occupation[by\_occupation.sum(1) > 1000000]  
  
**print '2008 over 1 million contributions'  
print** over\_1mm\_2008  
  
by\_occupation = fec\_2012.pivot(**'contb\_receipt\_amt'**, rows=**'contbr\_occupation'**,  
cols=**'party'**, aggfunc=**'sum'**)  
  
over\_1mm\_2012 = by\_occupation[by\_occupation.sum(1) > 1000000]  
  
**print '2012 over 1 million contributions'  
print** over\_1mm\_2012  
  
by\_occupation = fec\_2016.pivot(**'contb\_receipt\_amt'**, rows=**'contbr\_occupation'**,  
cols=**'party'**, aggfunc=**'sum'**)  
  
over\_1mm\_2016 = by\_occupation[by\_occupation.sum(1) > 1000000]  
  
**print '2016 over 1 million contributions'  
print** over\_1mm\_2016

1. **Calculating the donations by top 10 states**

*#Donations by state*grouped = fec.groupby([**'cand\_nm'**, **'contbr\_st'**])  
totals = grouped.contb\_receipt\_amt.sum().unstack(0).fillna(0)  
totals = totals[totals.sum(1) > 100000]  
**print** totals[:2]

cand\_nm Obama, Barack Paul

contbr\_st

AE 2.470553e+05

AK 1.042121e+06

AL 1.857058e+06

AP 9.166670e+04

AR 1.165400e+06

AZ 5.899544e+06

CA 1.043740e+08

CO 1.192509e+07

CT 1.062052e+07

DC 2.849802e+07

cand\_nm Romney, Mitt Tancredo,

contbr\_st

AE 3300.00

AK 51195.00

AL 313037.00

AP 3694.00

AR 35167.00

AZ 1764618.65

CA 10051002.99

CO 1089544.00

CT 1791204.18

DC 515346.05

Process finished with exit code 0

Validation method used – StratifiedKFold

It is a variation of *k-fold* which returns *stratified* folds: each set contains approximately the same percentage of samples of each target class as the complete set.

So we can see Obama has received highest donation from CA and Romey has received highest donation from AZ. So Romey tends to be more dominant in AZ and Obama is more dominant in CA.

1. Last feature I would consider while building the model is Zip code as the area where people live depict a bias towards the parties

4. Conduct a temporal analysis on the ability of your models to generalize. Discuss these results, and the results of your validation and form a prediction for each individual state in the 2016 election, along with the overall outcome for the nation. Provide suggestions to both parties for how to optimize their fund raising efforts based on your models of individual likelihood for contribution.

**I have generalized the model by taking the results from California State as it is a mixture of Democrats and Republican candidates. Republican wins in 35 states and Democrat wins in 45 states.**

**So according to my model 2016 election will be won by Democrat by 10 states.**

I have used the contribution receipt date as well to fit into the model and then conduct the temporal analysis.

Suggestions – The party can look at states where contribution to other party is higher like Arizona,it is heavily loaded by Democrat Retired contibutions. So the party can target other occupation candidates to contribute more for their party.

Bias in model – I have trained the model on 2008 and tested on 2012.As democrat was a winner in 2008, I believe the model is biased towards Democrats.

state = fec\_2016.contbr\_st.unique()  
**for** j **in** range(len(state)):  
 **for** i **in** range(len(fec\_2016[**"predict\_winner"**])) :  
 **if** fec\_2016.contbr\_st[i]==state[j]:  
 Democrat=0  
 Republican=0  
 Other=0  
 **if** fec\_2016.predict\_winner[i]==**'Democratic'**: *#and fec\_2016.contbr\_st[i]==state[j]:* Democrat=Democrat+1  
 **elif** fec\_2016.predict\_winner[i]==**'Republican'**: *#and fec\_2016.contbr\_st[i]==state[j]:* Republican=Republican+1  
 **else**:  
 Other=Other+1  
 **if** Democrat>Republican:  
  
 **print 'Democrat Wins in 2016 as per the Predictive Model for state'  
 print** state[j]  
 **else**:  
 **print 'Republican Wins in 2016 as per the Predictive Model for state'  
 print** state[j]

Below is the output of Republican and Democratic Wins state Wise-

Republican Wins in 2016 as per the Predictive Model for state

AE

Democrat Wins in 2016 as per the Predictive Model for state

AK

Republican Wins in 2016 as per the Predictive Model for state

AL

Republican Wins in 2016 as per the Predictive Model for state

AR

Republican Wins in 2016 as per the Predictive Model for state

AZ

Democrat Wins in 2016 as per the Predictive Model for state

CA

Democrat Wins in 2016 as per the Predictive Model for state

CO

Democrat Wins in 2016 as per the Predictive Model for state

CT

Democrat Wins in 2016 as per the Predictive Model for state

DC

Republican Wins in 2016 as per the Predictive Model for state

DE

Democrat Wins in 2016 as per the Predictive Model for state

FF

Republican Wins in 2016 as per the Predictive Model for state

FL

Republican Wins in 2016 as per the Predictive Model for state

GA

Democrat Wins in 2016 as per the Predictive Model for state

GU

Republican Wins in 2016 as per the Predictive Model for state

HI

Democrat Wins in 2016 as per the Predictive Model for state

IA

Democrat Wins in 2016 as per the Predictive Model for state

ID

Democrat Wins in 2016 as per the Predictive Model for state

IL

Democrat Wins in 2016 as per the Predictive Model for state

IN

Republican Wins in 2016 as per the Predictive Model for state

KS

Democrat Wins in 2016 as per the Predictive Model for state

KY

Democrat Wins in 2016 as per the Predictive Model for state

LA

Democrat Wins in 2016 as per the Predictive Model for state

MA

Republican Wins in 2016 as per the Predictive Model for state

MD

Democrat Wins in 2016 as per the Predictive Model for state

ME

Republican Wins in 2016 as per the Predictive Model for state

MI

Democrat Wins in 2016 as per the Predictive Model for state

MN

Democrat Wins in 2016 as per the Predictive Model for state

MO

Democrat Wins in 2016 as per the Predictive Model for state

MP

Democrat Wins in 2016 as per the Predictive Model for state

MS

Democrat Wins in 2016 as per the Predictive Model for state

MT

Democrat Wins in 2016 as per the Predictive Model for state

NC

Republican Wins in 2016 as per the Predictive Model for state

ND

Democrat Wins in 2016 as per the Predictive Model for state

NE

Democrat Wins in 2016 as per the Predictive Model for state

NH

Democrat Wins in 2016 as per the Predictive Model for state

NJ

Democrat Wins in 2016 as per the Predictive Model for state

NM

Democrat Wins in 2016 as per the Predictive Model for state

NV

Democrat Wins in 2016 as per the Predictive Model for state

NY

Republican Wins in 2016 as per the Predictive Model for state

OH

Democrat Wins in 2016 as per the Predictive Model for state

OK

Republican Wins in 2016 as per the Predictive Model for state

OR

Democrat Wins in 2016 as per the Predictive Model for state

PA

Republican Wins in 2016 as per the Predictive Model for state

PR

Democrat Wins in 2016 as per the Predictive Model for state

RI

Democrat Wins in 2016 as per the Predictive Model for state

SC

Democrat Wins in 2016 as per the Predictive Model for state

SD

Democrat Wins in 2016 as per the Predictive Model for state

SI

Republican Wins in 2016 as per the Predictive Model for state

TN

Republican Wins in 2016 as per the Predictive Model for state

TX

Republican Wins in 2016 as per the Predictive Model for state

UT

Republican Wins in 2016 as per the Predictive Model for state

VA

Republican Wins in 2016 as per the Predictive Model for state

VI

Democrat Wins in 2016 as per the Predictive Model for state

VT

Democrat Wins in 2016 as per the Predictive Model for state

WA

Republican Wins in 2016 as per the Predictive Model for state

WI

Democrat Wins in 2016 as per the Predictive Model for state

WV

Democrat Wins in 2016 as per the Predictive Model for state

WY

Republican Wins in 2016 as per the Predictive Model for state

XX

Democrat Wins in 2016 as per the Predictive Model for state

AP

Democrat Wins in 2016 as per the Predictive Model for state

AS

Democrat Wins in 2016 as per the Predictive Model for state

AU

Republican Wins in 2016 as per the Predictive Model for state

BC

Republican Wins in 2016 as per the Predictive Model for state

BR

Democrat Wins in 2016 as per the Predictive Model for state

LE

Democrat Wins in 2016 as per the Predictive Model for state

AA

Republican Wins in 2016 as per the Predictive Model for state

QC

Republican Wins in 2016 as per the Predictive Model for state

HO

Republican Wins in 2016 as per the Predictive Model for state

LO

Republican Wins in 2016 as per the Predictive Model for state

MB

Republican Wins in 2016 as per the Predictive Model for state

NL

Republican Wins in 2016 as per the Predictive Model for state

NS

Democrat Wins in 2016 as per the Predictive Model for state

ON

Republican Wins in 2016 as per the Predictive Model for state

TO

Democrat Wins in 2016 as per the Predictive Model for state

BU

Democrat Wins in 2016 as per the Predictive Model for state

GE

Republican Wins in 2016 as per the Predictive Model for state

N.

Democrat Wins in 2016 as per the Predictive Model for state

NO

Republican Wins in 2016 as per the Predictive Model for state

C

Republican Wins in 2016 as per the Predictive Model for state

ZZ

**CODE**

\_\_author\_\_ = **'charukhatwani'***#Array processing***import** subprocess  
**import** numpy **as** np  
  
**from** sklearn **import** datasets  
**from** sklearn **import** cross\_validation  
**from** sklearn **import** preprocessing  
**from** sklearn.metrics **import** confusion\_matrix,accuracy\_score,classification\_report  
**from** sklearn.cross\_validation **import** StratifiedKFold  
*#Data analysis, wrangling and common exploratory operations***import** pandas **as** pd  
**from** pandas **import** Series, DataFrame  
  
*#For visualization. Matplotlib for basic viz and seaborn for more stylish figures + statistical figures not in MPL.***import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sns  
  
*# Decision tree classifier***from** sklearn.tree **import** DecisionTreeClassifier, export\_graphviz  
**from** sklearn.ensemble **import** RandomForestClassifier  
  
**import** pandas **as** pd  
  
  
fec = pd.read\_csv(**'P00000001-OH-2008.csv'**,index\_col=False)  
fec\_2012=pd.read\_csv(**'P00000001-OH-2012.csv'**,index\_col=False)  
fec\_2016=pd.read\_csv(**'P00000001-ALL-2016.csv'**,index\_col=False)  
**print** fec.info()  
  
unique\_cands\_2008= fec.cand\_nm.unique()  
unique\_cands\_2012= fec\_2012.cand\_nm.unique()  
unique\_cands\_2016= fec\_2016.cand\_nm.unique()  
  
**print '2008 candidates'  
print** unique\_cands\_2008  
**print '2012 candidates'  
print** unique\_cands\_2012  
**print '2016 candidates'  
print** unique\_cands\_2016  
  
  
*#while doing analysis later we want to associate each candidate with their party*parties = {**'Bachmann, Michele'**: **'Republican'**,  
 **'Cain, Herman'**: **'Republican'**,  
 **'Gingrich, Newt'**: **'Republican'**,  
 **'Huntsman, Jon'**: **'Republican'**,  
 **'Johnson, Gary Earl'**: **'Republican'**,  
 **'McCotter, Thaddeus G'**: **'Republican'**,  
 **'Obama, Barack'**: **'Democratic'**,  
 **'Paul, Ron'**: **'Republican'**,  
 **'Pawlenty, Timothy'**: **'Republican'**,  
 **'Perry, Rick'**: **'Republican'**,  
 **"Roemer, Charles E. 'Buddy' III"**: **'Republican'**,  
 **'Romney, Mitt'**: **'Republican'**,  
 **'Santorum, Rick'**: **'Republican'**,  
 **'Stein, Jill'**:**'Green'**,  
 **'Rubio, Marco'**:**'Republican'**,  
 **'Santorum, Richard J.'**:**'Republican'**,  
 **'Perry, James R. (Rick)'**:**'Republican'**,  
 **'Carson, Benjamin S.'**:**'Republican'**,  
 **"Cruz, Rafael Edward 'Ted'"**:**'Republican'**,  
 **'Paul, Rand'**:**'Republican'**,  
 **'Clinton, Hillary Rodham'**:**'Democratic'**,  
 **'Sanders, Bernard'**:**'Democratic'**,  
 **'Fiorina, Carly'**:**'Republican'**,  
 **'Huckabee, Mike'**:**'Republican'**,  
 **'Pataki, George E.'**:**'Republican'**,  
 **"O'Malley, Martin Joseph"**:**'Democratic'**,  
 **'Graham, Lindsey O.'**:**'Republican'**,  
 **'Bush, Jeb'**:**'Republican'**,  
 **'Trump, Donald J.'**:**'Republican'**,  
 **'Jindal, Bobby'**:**'Republican'**,  
 **'Christie, Christopher J.'**:**'Republican'**,  
 **'Walker, Scott'**:**'Republican'**,  
 **'Webb, James Henry Jr.'**:**'Republican'**,  
 **'Kasich, John R.'**:**'Republican'**,  
 **'Lessig, Lawrence'**:**'Democratic'**,  
 **'Huckabee, Mike'**: **'Republican'**,  
 **'Paul, Ron'**: **'Republican'**,  
 **'Hunter, Duncan'**: **'Republican'**,  
 **'Thompson, Fred Dalton'**: **'Republican'**,  
 **'Richardson, Bill'**: **'Democratic'**,  
 **'McCain, John S'**: **'Republican'**,  
 **'Clinton, Hillary Rodham'**: **'Democratic'**,  
 **'Edwards, John'**: **'Democratic'**,  
 **'Giuliani, Rudolph W'**: **'Republican'**,  
 **'Brownback, Samuel Dale'**:**'Republican'**,  
 **'Tancredo, Thomas Gerald'**: **'Republican'**,  
 **'Cox, John H'**:**'Republican'**,  
 **'Biden, Joseph R Jr'**: **'Democratic'**,  
 **'Gravel, Mike'**: **'Democratic'**,  
 **'Dodd, Christopher J'**:**'Democratic'**,  
 **'Kucinich, Dennis J'**:**'Democratic'**,  
 **'Gilmore, James S III'**:**'Republican'**,  
 **''**:**'Other'**}  
  
*# Adding a column party that sets value to the party candidates  
# The way this line works is as follows:  
# 1. fec\_all.cand\_nm gives a vector (or Series in Pandas terminology)  
# 2. For each row, the code looks up the candidate name to the dictionary parties  
# 3. If the name of the candidate (cand\_nm) is in parties, it returns the value (i.e. Republican or Democrat)  
# 4. This whole thing is done for each row and you get another vector as output  
# 5. Finally, we create a new column in fec\_all called 'party' and assign the vector  
#print fec.cand\_nm[123456:123461]*fec[**'party'**] = fec.cand\_nm.map(parties)  
fec\_2012[**'party'**] = fec\_2012.cand\_nm.map(parties)  
fec\_2016[**'party'**] = fec\_2016.cand\_nm.map(parties)  
  
*#Calculating the counts of each year party wise***print '2008 party wise vote counts'  
print** fec[**'party'**].value\_counts()  
  
**print '2012 party wise vote counts'  
print** fec\_2012[**'party'**].value\_counts()  
  
**print '2016 party wise vote counts'  
print** fec\_2016[**'party'**].value\_counts()  
  
*#To simplify analysis I will restrict only to positive contributions*fec = fec[fec.contb\_receipt\_amt > 0]  
fec\_2012 = fec\_2012[fec\_2012.contb\_receipt\_amt > 0]  
fec\_2016 = fec\_2016[fec\_2016.contb\_receipt\_amt > 0]  
  
fec\_2012[**'party'**] = fec\_2012.cand\_nm.map(parties)  
fec\_2016[**'party'**] = fec\_2016.cand\_nm.map(parties)  
  
  
*#Top 10 Contribution Statistics by Occupation*Contribution\_2008=fec.contbr\_occupation.value\_counts()[:10]  
**print '2008 top 10 contributions by occupation'  
print** Contribution\_2008  
  
  
Contribution\_2012=fec\_2012.contbr\_occupation.value\_counts()[:10]  
**print '2012 top 10 contributions by occupation'  
print** Contribution\_2012  
  
  
Contribution\_2016=fec\_2016.contbr\_occupation.value\_counts()[:10]  
**print '2016 top 10 contributions by occupation'  
print** Contribution\_2016  
  
*#Now we see the bifurcation donation party wise  
#import pandas.util.testing as tm; tm.N = 3*by\_occupation = fec.pivot\_table(**'contb\_receipt\_amt'**, index=**'contbr\_occupation'**,  
columns=**'party'**,aggfunc=np.sum)  
  
over\_1mm\_2008 = by\_occupation[by\_occupation.sum(1) > 1000000]  
  
**print '2008 over 1 million contributions'  
print** over\_1mm\_2008  
  
by\_occupation = fec\_2012.pivot\_table(**'contb\_receipt\_amt'**, index=**'contbr\_occupation'**,  
columns=**'party'**,aggfunc=np.sum)  
  
over\_1mm\_2012 = by\_occupation[by\_occupation.sum(1) > 1000000]  
  
**print '2012 over 1 million contributions'  
print** over\_1mm\_2012  
  
by\_occupation = fec\_2016.pivot\_table(**'contb\_receipt\_amt'**, index=**'contbr\_occupation'**,  
columns=**'party'**,aggfunc=np.sum)  
  
  
  
over\_1mm\_2016 = by\_occupation[by\_occupation.sum(1) > 1000000]  
  
**print '2016 over 1 million contributions'  
print** over\_1mm\_2016  
  
  
**def** get\_top\_amounts(group, key, n=5):  
 totals = group.groupby(key)[**'contb\_receipt\_amt'**].sum()  
*# Order totals by key in descending order return totals.order(ascending=False)[-n:]  
  
#Calculating the top 10 occupations*grouped = fec.groupby(**'cand\_nm'**)  
**print** grouped.apply(get\_top\_amounts, **'contbr\_occupation'**, n=5)  
  
*#Donations by state*grouped = fec.groupby([**'cand\_nm'**, **'contbr\_st'**])  
totals = grouped.contb\_receipt\_amt.sum().unstack(0).fillna(0)  
totals = totals[totals.sum(1) > 100000]  
**print** totals[:10]  
  
by\_occupation = fec.pivot\_table(**'contb\_receipt\_amt'**, index=**'contbr\_zip'**,  
columns=**'party'**,aggfunc=np.sum)  
  
over\_1mm\_2008 = by\_occupation[by\_occupation.sum(1) > 1000000]  
  
**print** over\_1mm\_2008  
  
*#We tend to use zip code as a categorial variable and trimming the zip value to contain only 5 characters*fec[**"zip\_valid"**]=(fec.contbr\_zip).map(**lambda** x:str(x)[:5])  
fec[**"zip\_valid"**]=(fec.contbr\_zip).map({**'NG7 1'**: 12345,**'V6R 3M9'**: 12345,**'M5T 2'**: 12345,**'4221s'**:12345})  
**print** fec.zip\_valid[fec.zip\_valid.isin([**'M5T 2'**])]  
  
fec[**"zip\_valid"**]=pd.DataFrame(fec[**"zip\_valid"**].astype(float))  
fec[**"zip\_valid"**]=fec[**"zip\_valid"**].fillna(value=0)  
  
  
fec\_2012[**"zip\_valid"**]=(fec\_2012.contbr\_zip).map(**lambda** x:str(x)[:5])  
fec\_2012[**"zip\_valid"**]=(fec\_2012.contbr\_zip).map({**'NG7 1'**: 12345,**'V6R 3M9'**: 12345,**'M5T 2'**: 12345,**'4221s'**:12345})  
  
fec\_2012[**"zip\_valid"**]=pd.DataFrame(fec\_2012[**"zip\_valid"**].astype(float))  
fec\_2012[**"zip\_valid"**]=fec\_2012[**"zip\_valid"**].fillna(value=0)  
  
fec\_2016[**"zip\_valid"**]=(fec\_2016.contbr\_zip).map(**lambda** x:str(x)[:5])  
fec\_2016[**"zip\_valid"**]=(fec\_2016.contbr\_zip).map({**'NG7 1'**: 12345,**'V6R 3M9'**: 12345,**'M5T 2'**: 12345,**'4221s'**:12345})  
  
fec\_2016[**"zip\_valid"**]=pd.DataFrame(fec\_2016[**"zip\_valid"**].astype(float))  
fec\_2016[**"zip\_valid"**]=fec\_2016[**"zip\_valid"**].fillna(value=0)  
  
  
*#Random forest treat int values as categorial  
  
  
#using occupation as a feature*temp\_fec = pd.DataFrame({**'contbr\_occupation'**: fec.contbr\_occupation.unique(), **'contbr\_occupation\_id'**:range(len(fec.contbr\_occupation.unique()))})  
fec = fec.merge(temp\_fec, on=**'contbr\_occupation'**, how=**'left'**)  
  
  
*# Adding occupation id column for occupation names 2012*temp\_fec\_2012 = pd.DataFrame({**'contbr\_occupation'**: fec\_2012.contbr\_occupation.unique(), **'contbr\_occupation\_id'**:range(len(fec\_2012.contbr\_occupation.unique()))})  
fec\_2012 = fec\_2012.merge(temp\_fec\_2012, on=**'contbr\_occupation'**, how=**'left'**)  
  
  
  
*# Adding occupation id column for occupation names 2016*temp\_fec\_2016 = pd.DataFrame({**'contbr\_occupation'**: fec\_2016.contbr\_occupation.unique(), **'contbr\_occupation\_id'**:range(len(fec\_2016.contbr\_occupation.unique()))})  
fec\_2016 = fec\_2016.merge(temp\_fec\_2016, on=**'contbr\_occupation'**, how=**'left'**)  
  
  
TrainingFeatures = fec[[**'contb\_receipt\_amt'**,**'zip\_valid'**,**'contbr\_occupation\_id'**]]  
TrainingClassLabels = fec[**'party'**]  
classifier = RandomForestClassifier(n\_estimators=100)  
clf = classifier.fit(TrainingFeatures,TrainingClassLabels)  
  
**print** classifier.max\_features  
*#Testing on 2012 Data*TestFeatures = fec\_2012[[**'contb\_receipt\_amt'**,**'zip\_valid'**,**'contbr\_occupation\_id'**]]  
TestClassLabels=fec\_2012[**'party'**]  
Test\_predict=clf.predict(TestFeatures)  
  
*#Cross Validating on 2012 data*ValidationFeatures = fec\_2012[[**'contb\_receipt\_amt'**,**'zip\_valid'**,**'contbr\_occupation\_id'**]]  
ValidationClassLabels=fec\_2012[**'party'**]  
  
scores=cross\_validation.cross\_val\_score(clf,ValidationFeatures,ValidationClassLabels,cv=5)  
**print 'Scores'  
print** scores  
**print**(**"Accuracy: %0.2f (+/- %0.2f)"** % (scores.mean(), scores.std() \* 2))  
  
*#K fold cross validation*labels = fec\_2012.party  
  
skf=StratifiedKFold(labels,3)  
**for** train,test **in** skf:  
 **print**(**"%s %s"**%(train,test))  
  
  
*#Now we will predict 2016 results*PredictFeatures = fec\_2016[[**'contb\_receipt\_amt'**,**'zip\_valid'**,**'contbr\_occupation\_id'**]]  
*#TestClassLabels=fec\_2012['party']*Predict2016 = clf.predict(PredictFeatures)  
  
**print** len(Predict2016)  
fec\_2016[**"predict\_winner"**] = Predict2016  
  
*#For each state, predict winner of popular vote.  
  
#for i in range(len())***print '2016 results'**Final\_Democrat=0  
Final\_Republican=0  
state = fec\_2016.contbr\_st.unique()  
**for** j **in** range(len(state)):  
 **for** i **in** range(len(fec\_2016[**"predict\_winner"**])) :  
 **if** fec\_2016.contbr\_st[i]==state[j]:  
 Democrat=0  
 Republican=0  
 Other=0  
 **if** fec\_2016.predict\_winner[i]==**'Democratic'**: *#and fec\_2016.contbr\_st[i]==state[j]:* Democrat=Democrat+1  
 **elif** fec\_2016.predict\_winner[i]==**'Republican'**: *#and fec\_2016.contbr\_st[i]==state[j]:* Republican=Republican+1  
 **else**:  
 Other=Other+1  
 **if** Democrat>Republican:  
  
 **print 'Democrat Wins in 2016 as per the Predictive Model for state'  
 print** state[j]  
 Final\_Democrat=Final\_Democrat+1  
 **else**:  
 **print 'Republican Wins in 2016 as per the Predictive Model for state'  
 print** state[j]  
 Final\_Republican=Final\_Republican+1  
  
  
**print 'Democrat wins in states'  
print** Final\_Democrat  
  
**print 'Republican wins in states'  
print** Final\_Republican  
  
  
  
  
  
  
  
cm=confusion\_matrix(TestClassLabels,Test\_predict)  
*#print cm*np.set\_printoptions(precision=2)  
**print**(**'Confusion matrix ,without normalization'**)  
**print**(cm)  
**print 'Accuracy'  
print** accuracy\_score(TestClassLabels, Test\_predict, normalize=False)  
  
labels=[**'Democratic'**,**'Republicans'**]  
*#plt.matshow(cm)  
#plt.title('Normalized Confusion matrix')  
#plt.colorbar()*fig = plt.figure()  
ax = fig.add\_subplot(111)  
cax = ax.matshow(cm)  
plt.title(**'Normalized Confusion matrix of the classifier'**)  
fig.colorbar(cax)  
ax.set\_xticklabels([**''**] + labels)  
ax.set\_yticklabels([**''**] + labels)  
plt.ylabel(**'True label'**)  
plt.xlabel(**'Predicted label'**)  
plt.show()  
  
  
*#print clf.feature\_importances\_*cm\_normalized=cm.astype(**'float'**)/cm.sum(axis=1)[:,np.newaxis]  
**print**(**'Normalized Confusion Matrix'**)  
**print**(cm\_normalized)  
**print 'Normalized Accuracy'  
print** accuracy\_score(TestClassLabels, Test\_predict)  
labels=[**'Democratic'**,**'Republicans'**]  
*#plt.matshow(cm)  
#plt.title('Normalized Confusion matrix')  
#plt.colorbar()*fig = plt.figure()  
ax = fig.add\_subplot(111)  
cax = ax.matshow(cm\_normalized)  
plt.title(**'Normalized Confusion matrix of the classifier'**)  
fig.colorbar(cax)  
ax.set\_xticklabels([**''**] + labels)  
ax.set\_yticklabels([**''**] + labels)  
plt.ylabel(**'True label'**)  
plt.xlabel(**'Predicted label'**)  
plt.show()  
target\_names = [**'class 0'**, **'class 1'**,**'class 2'**]  
  
**print**(classification\_report(TestClassLabels, Test\_predict, target\_names=target\_names))