Data Inspection & Cleaning

Before I start the machine learning, I did data inspection and cleaning.

From this survey result, the data set shows me that D2007, D2008, D2009 have similar value and same order, which is not meaningful for the research analysis.

```
for column in data:
    print(column)
    print(data[column].value_counts())
```

Before the machine Learning, I would like to clean the dataset first for getting rid of insignificant data such as 'unnamed: 0' and 'D2005' to 'D2009', so I dropped out unnamed:0 and D2005 to D2008 by keeping D2009.

There was no missing value.

```
print(data.isnull().sum())
```

And I did one-hot encode, using `pd.get_dummies()`

Too many unique values by 2016?

After this, I changed the categorical label to numeric label, which is 'voted' column for better performance, because some model types cannot perform with non-numerical data. So I label encoded those non-numerical columns using `sklearn LabelEncoder`.

```
from sklearn.preprocessing import LabelEncoder
label_voted = LabelEncoder()
data['voted'] = label_voted.fit_transform(data['voted'])
```

Variable selection

Setting the dependent and independent variables:

```
Y = new_data['voted']
X= new_data.drop(columns = ['voted'], axis=1)
```

Train-Test Split

Set the dependent and independent variables and split train and test

The dataset divided into: a training set and a testing set. 80% for training, and 20% for testing, so that I trained the model using the training set and tested the model using the testing set.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
random_state = 0)
```

Optimization

Scaling the data to take account of variations in mean and standard deviations

```
sc=StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

Multiple approaches with 5 different models

I chose the 5 different models:

1. Linear Regression
2. K-Neighbors Classifier Method
3. SVC Method
4. Decision Tree Model
5. Random Forest

Especially SVC model and Random Forest took almost 3-4 hours for each to train the dataset due to huge data set with my current laptop setting. I was able to successfully run the test examples, but on executing using my dataset and letting it run for over hours, I could hardly see output or termination of the program. I tried executing using (gamma='auto', kernel = 'linear) by SVC. SVC requires the computation of a distance function between each point in the dataset, which is the dominating cost of O. The storage of the distances is a burden on memory, so they're recomputed on the fly.

Accuracy Assessment

```
[0]Logistic Regression Training Accuracy: 0.8615
[0]Logistic Regression AUC Score : 0.641790

[1]K Nearest Neighbor Training Accuracy: 0.8575
[1]K Nearest Neighbor AUC Score : 0.665743

[2]Support Vector Machine (Linear Classifier) Training Accuracy: 0.8615
[2]Support Vector Machine (Linear Classifier) AUC Score : 0.610553

[3]Decision Tree Classifier Training Accuracy: 0.7961
[3]Decision Tree Classifier AUC Score : 0.650439

[4]Random Forest Classifier Training Accuracy: 0.8699
[4]Random Forest Classifier AUC Score : 0.633172
```

As executing models, I got at first time extremely high estimated of accuracy 1,0 by using Decision Tree Model. For avoiding this overfitting, I tried to come up with n_estimators and learning rate and tune other parameters. And then I could get proper accuracy like above.

As you can see from the above results, the accuracy can be increased by choosing the right model.

Optimization

So that I wanted to choose the random forest model for optimization and random search. Due to lack of my laptop setting, I was unfortunately not able to run the Random Forest model over hours, I could hardly see output or termination of the program, which was too heavy for my laptop CPU. So I chose K-Nearest Neighbor Model for Optimization with StandardScaler.

```
[1]K Nearest Neighbor Training Accuracy: 0.8226
[1]K Nearest Neighbor AUC Score : 0.578450
```

As you can see, the accuracy decreases. I assume that the data set probably doesn’t need to have the features at the same scale for some reason (or they are already at comparable scales and the normalization screwed it up for some reason).

Afterwards, I would like to try Random Search. Due to lack of laptop setting, I was not able to run more code in my laptop. But I attached my coding part for random search in .py file, so that the readerer can try at least by themselves.

Extra: Best Features

I applied 'SelectKBest' to extract top best 10 features:

```
bestfeatures = SelectKBest(score_func=chi2, k=15)
```

The best feature was age which has absolute highest feature score around 3763, so that I can conclude that 'age' is most influent independent variable of data set for election. And 'D2004_4' and D2004_1 come as second and third important independent variables.

	Specs	Score
0	age	3763.177437
19	D2004_4	439.346786
16	D2004_1	110.440446
9	D2003_7	42.992661
6	D2003_4	27.951370
5	D2003_3	21.047711
10	D2003_8	19.496158
4	D2003_2	7.214727
12	D2003_96	6.199791
17	D2004_2	5.047920
7	D2003_5	3.721072
13	D2003_97	2.278766
11	D2003_9	1.657446
8	D2003_6	1.487155
15	D2003_99	1.431323