2016 US Presidential Election Results Analysis

Name of author: Hans Peter Ndeffo

Department: Mathematics

Date: 3/23/2019

Summary

The results of US 2016 presidential election surprised many people throughout the w
orld. Hillary Clinton was the favorite candidate with potentially the

highest number of voters based on preliminary analysis. Therefore, how can Donald T
rump can outclass all the predictions against him and win? The purpose of

this report is, consequently, to establish if the outcome of the election could be
 predicted using regressions models and statiscal techniques.

Procedure:

Our analysis will be split into 7 steps:

 Getting 10 last US presidential election votes the including US presidential election.

 Defining the model, variables and dependencies between if there is.

 Looking for the best model(linear, polynomial regression) that can be used to predict the 2016 US presidential election. What is important to note is that the more US presidential elections included in the analysis

the better the prediction should be. Also the choice of the best model will be based on the split and train technic and further details will be given later

 Getting the US 2016 presidential election tally per states

 Pre-process the data using numpy, sklearn to check for empty rows or columns.

 Sorting the data using R studio to get only the last digit from 0 to 9 of the total number of voters of each states and count how many times they appear.

 Use the Chi-Squared to see if the vote tally is uniformly distributed based on the previous elections distribution. That will determine if there is a fraud or not with confidence level of 95%

 Conclude

Contents

In [2]:
```
# Import the libraries
import numpy as np
import pandas as pd
from sklearn import svm
import matplotlib.pyplot as plt
```

An excel file or table that has all the us presidential elections will be imported

Lets have a look at the table

In [3]:
```
dataset0 = pd.read_csv("US Elections Result.csv")
dataset0.iloc[:,0:-1]
```

Out[3]:

	Election year	Main Candidates	Main Party	Popular Votes
0	2016	Donald J. Trump	Republican	62980160
1	2016	Hillary R. Clinton	Democratic	65845063
2	2012	W. Mitt Romney	Republican	60589084
3	2012	Barack H. Obama	Democratic	65446032
4	2008	John S. McCain	Republican	59934814
5	2008	Barack H. Obama	Democratic	69456897
6	2004	George W. Bush	Republican	62039073
7	2004	John F. Kerry	Democratic	59027478
8	2000	George W. Bush	Republican	50456062
9	2000	Albert Gore, Jr.	Democratic	50996582
10	1996	Robert Dole	Republican	37816307
11	1996	William J. Clinton	Democratic	45590703
12	1992	George Bush	Republican	39102343
13	1992	William J. Clinton	Democratic	44908254
14	1988	George Bush	Republican	47946000
15	1988	Michael S. Dukakis	Democratic	41016000
16	1984	Ronald Reagan	Republican	54455000
17	1984	Walter F. Mondale	Democratic	37577000
18	1980	Ronald Reagan	Republican	43901812
19	1980	Jimmy Carter	Democratic	35483820
20	1976	Gerald R. Ford	Republican	40825839
21	1976	Jimmy Carter	Democratic	39147770

Defining the model

Assumptions:

-The number of votes strongly depends on the states and the number of people that have voted that year in that state

-

Variables:

- Number of votes obtain in a year for a given state: It is a variable because it is not known it is what this analysis aims to predict

However in order to make a prediction the results of one election is not enough because it does not provide information to any model, such as SVM regression.

Therefore it is reasonable to think in order to make a better prediction it is required to have as many election results for republicans and democrats.

Thus the more election results are involved the better the prediction should be.

So the next part will be to import those data and use some of them to train our model.

The goal is to predict who will win the election estimating the number of votes for republicans and democrates for every 4 years

In [4]:
```
#Create a list that contains the elections year that will be used for the plot
years = []

for i in range(11):
    years.append(1976 + 4*i)

# Display the list
years
```

Out[4]: [1976, 1980, 1984, 1988, 1992, 1996, 2000, 2004, 2008, 2012, 2016]

In [5]:
```
dataset0.iloc[2*0:2*0+1,3].values[0]
```

Out[5]: 62980160

In [6]:
```
#NoOfVotes = dataset0.iloc[:,3].values

# Number of votes for the republican candidate
repNoOfVotes = []
# Number of votes for the democrate candidate
demNoOfVotes = []

# Get the number of votes from the republican candidate according to the table above
for i in range(len(years)):
    # Append the corresponding vote to the appropriate candidate
    repNoOfVotes.append(dataset0.iloc[2*i:2*i+1,3].values[0])
    demNoOfVotes.append(dataset0.iloc[2*i+1:2*i+2,3].values[0])
```

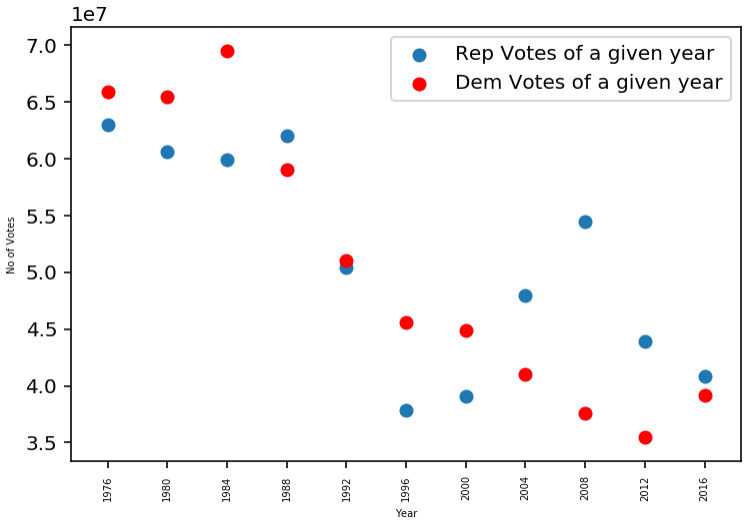
In [7]:
```
# indexes will be generates to plots the states names on the x-axis
index0 = np.arange(len(years))
index0
```

Out[7]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [8]:
```
p1 = plt.scatter(index0,repNoOfVotes,label="Rep Votes of a given year")
p2 = plt.scatter(index0,demNoOfVotes,color = 'red',label="Dem Votes of a given year")
#plt.bar(index0,repNoOfVotes)
plt.xlabel('Year',fontsize=5)
plt.ylabel('No of Votes',fontsize=5)
plt.xticks(index0,years, fontsize = 5, rotation=90)
plt.legend(loc="best")

plt.show()
```

Out[8]:



Crossing out the regressions models that can not be used

- Just by looking on how the data are spread it can be seen it is not possible to draw a line to fit Republicans nor democrates votes
- A logistic model can be used also because the numbers is not a constant state such true or false although a each presidential election has only 2 outcomes(Is a democrate elected or a republican elected every 4 years or not)

Therefore the first regression to be tested will be the SVM regression

In [9]:
```
# An array that has all the elections years represented in order by indexes
votingYears = []
for i in range (11):
    nextVotingYear = [i]
    votingYears.append(nextVotingYear)
votingYears
```

Out[9]:
```
[[0], [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]]
```

In [10]:
```
X = np.array(votingYears)
y = np.array(demNoOfVotes)
yPredict = svm.SVR().fit(X, y).predict(X)
yPredict
```

```
/ext/anaconda5/lib/python3.6/site-packages/sklearn/svm/base.py:196: FutureWar
ning: The default value of gamma will change from 'auto' to 'scale' in versio
n 0.22 to account better for unscaled features. Set gamma explicitly to 'aut
o' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[10]:
```
array([45590704.3863186 , 45590704.75419793, 45590704.77239005,
       45590704.75407441, 45590704.36787944, 45590703.        ,
       45590701.63212056, 45590701.24592559, 45590701.22760995,
       45590701.24580207, 45590701.6136814 ])
```

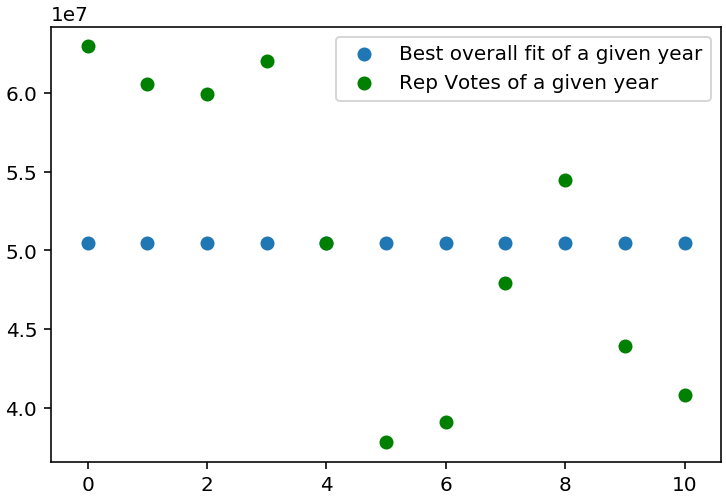
In [11]:
```
demNoOfVotes
```

Out[11]:
```
[65845063,
 65446032,
 69456897,
 59027478,
 50996582,
 45590703,
 44908254,
 41016000,
 37577000,
 35483820,
 39147770]
```

In [12]:
```
X = np.array(votingYears)
y = np.array(repNoOfVotes)
svr_rbf = svm.SVR(kernel='rbf', C=1e3, gamma=0.1)
yPredict = svr_rbf.fit(X, y) .predict(X)
p3 = plt.scatter(index0,yPredict,label="Best overall fit of a given year")
p4 = plt.scatter(index0,repNoOfVotes, color = 'green',label="Rep Votes of a gi
ven year")
plt.legend(loc="best")
#plt.show()
yPredict
```

Out[12]:
```
array([50458633.80012227, 50458941.78681409, 50458575.27497225,
       50457517.87127753, 50456061.99999931, 50454675.91758583,
       50453737.66961543, 50453339.15775471, 50453329.24795792,
       50453525.56486255, 50453852.00421544])
```

Out[12]:



In [13]:
```
# Import train_test_split function
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(index0, repNoOfVotes, test
_size=0.3,random_state=40) # 70% of data will be use for training and 30% for
 testing
```

This simply means that a part of the data will be used to predict the outcome of the 30%

remaining. This technique is commonly used to predict data based on the others. The X train, Xtest and Y train, Y test are split the same the way as shown as below. However the size

of the training data depends and may vary from 1 testing to another one. However if it is

possible to predict outcomes from testing data from a really small training set, it means that

the model that has been built is relatively robust.

In [14]:
```
# Each number such as 1 stands for 1980, 2 for 1984
X_train
```

Out[14]:
```
array([1, 2, 9, 0, 5, 7, 6])
```

In [15]:
```
X_test
```

Out[15]:
```
array([ 3,  4,  8, 10])
```

In [16]:
```
y_train
```

Out[16]:
```
[60589084, 59934814, 43901812, 62980160, 37816307, 47946000, 39102343]
```

In [17]:
```
y_test
```

Out[17]:
```
[62039073, 50456062, 54455000, 40825839]
```

In [45]:
```
for i in range (len(repNoOfVotes)):
    print(str(repNoOfVotes[i]) + " with an index " + str(i) + " which represen
ts the number of votes for the year " + str(years[i]))
```

```
62980160 with an index 0 which represents the number of votes for the year 19
76
60589084 with an index 1 which represents the number of votes for the year 19
80
59934814 with an index 2 which represents the number of votes for the year 19
84
62039073 with an index 3 which represents the number of votes for the year 19
88
50456062 with an index 4 which represents the number of votes for the year 19
92
37816307 with an index 5 which represents the number of votes for the year 19
96
39102343 with an index 6 which represents the number of votes for the year 20
00
47946000 with an index 7 which represents the number of votes for the year 20
04
54455000 with an index 8 which represents the number of votes for the year 20
08
43901812 with an index 9 which represents the number of votes for the year 20
12
40825839 with an index 10 which represents the number of votes for the year 2
016
```

So the numbers such as 3,4,8 in X test represents the year 1980,1984,2012 whose matching votes are represented in the cell above. The next step will be to create an array that will be used to compute the svm function

In [38]:
```
# Creating an array for the X train set
votingYearsTrain = []
for i in range (len(X_train)):
    # Append the voting year corresponding to i

     nextVotingYearTrain = [i]
     votingYearsTrain.append([X_train[i]])
votingYearsTrain
```

Out[38]:
```
[[1], [2], [9], [0], [5], [7], [6]]
```

In [39]:
```
# Creating an array for the X test set
votingYearsTest = []
for i in range (len(X_test)):
   # Append the voting year corresponding to i
    votingYearsTest.append([X_test[i]])
votingYearsTest
```

Out[39]:
```
[[3], [4], [8], [10]]
```

Now the variable **votingYearsTrain** is **ready to be used in the svm function**

In [40]:
```
# Creating an array for the X train set
repVotingTrainSet = []
for i in range (len(y_train)):
    # Append the voting year corresponding to i
    repVotingTrainSet.append( repNoOfVotes[votingYearsTrain[i][0]])
repVotingTrainSet
```

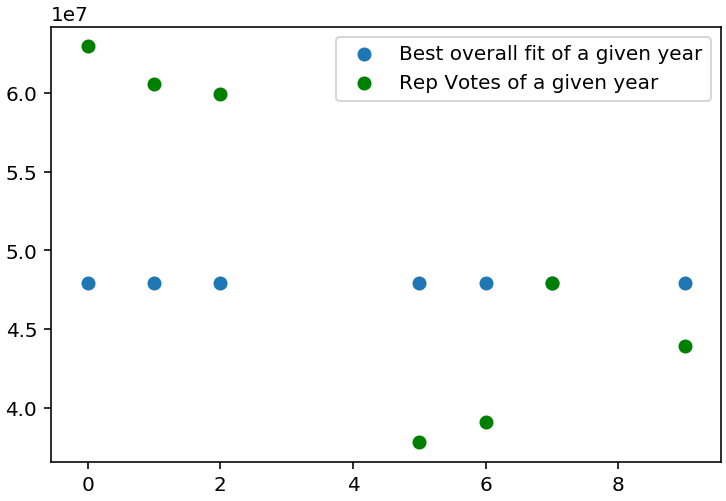
Out[40]:
```
[60589084, 59934814, 43901812, 62980160, 37816307, 47946000, 39102343]
```

Now the variable **repVotingTrainSet** is **ready to be used in the svm function**. The efficiency of the SVM will be tested to see if it fits the data that have been used to train for this model.

In [41]:
```
X = np.array(votingYearsTrain)
y = np.array(repVotingTrainSet)
svr_rbf = svm.SVR(kernel='rbf', C=1e3, gamma=0.1)
yPredict = svr_rbf.fit(X, y) .predict(X)
p3 = plt.scatter(votingYearsTrain,yPredict,label="Best overall fit of a given
 year")
p4 = plt.scatter(votingYearsTrain,repVotingTrainSet, color = 'green',label="Re
p Votes of a given year")
plt.legend(loc="best")
#plt.show()
yPredict
```

Out[41]:
```
array([47950652.65395554, 47950087.86689662, 47946529.5676951 ,
       47950594.06739722, 47946712.43943369, 47945999.99998739,
       47946128.52035465])
```

Out[41]:



In [43]:
```
#Those are the predicted votes for the states base on the SVM model
svr_rbf.fit(X, y) .predict(np.array(votingYearsTest))
```

Out[43]:
```
array([47949006.13588895, 47947750.16595443, 47946183.32693234,
       47946941.81375504])
```

Actual results are:

In [44]:
```
y_test
```

Out[44]:
```
[62039073, 50456062, 54455000, 40825839]
```

The prediction is not relatively close overall because the percent difference is more than 10% overall. So the next model to be tested will be the K-NN regression

Verifying the assumptions before using the K-NN regression

In [62]:
```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski')
classifier.fit(np.array(votingYearsTrain),y_train)
classifier.predict(np.array(votingYearsTest))
```

Out[62]:
```
array([37816307, 37816307, 37816307, 37816307])
```

In [0]:
```
X = np.array(votingYearsTrain)
y = np.array(repVotingTrainSet)
svr_rbf = svm.SVR(kernel='rbf', C=1e3, gamma=0.1)
yPredict = svr_rbf.fit(X, y) .predict(X)
p3 = plt.scatter(votingYearsTrain,yPredict,label="Best overall fit of a given
 year")
p4 = plt.scatter(votingYearsTrain,repVotingTrainSet, color = 'green',label="Re
p Votes of a given year")
plt.legend(loc="best")
#plt.show()
yPredict
```

In [0]:
```
X = np.array(votingYearsTrain)
y = np.array(repVotingTrainSet)
svr_rbf = svm.SVR(kernel='rbf', C=1e3, gamma=0.1)
yPredict = svr_rbf.fit(X, y) .predict(X)
p3 = plt.scatter(votingYearsTrain,yPredict,label="Best overall fit of a given
 year")
p4 = plt.scatter(votingYearsTrain,repVotingTrainSet, color = 'green',label="Re
p Votes of a given year")
plt.legend(loc="best")
#plt.show()
yPredict
```

In [42]:
```
dataset = pd.read_csv("nytimes_presidential_elections_2016_results_county.csv"
)
dataset
```

Out[42]:

	Clinton	Trump	Rpt	State	Vote by county	Vote by town	Place
0	151581.0	130614.0	100	Alabama	Jefferson	NaN	Jefferson
1	68429.0	91087.0	100	Alabama	Mobile	NaN	Mobile
2	62435.0	89199.0	100	Alabama	Madison	NaN	Madison
3	22927.0	72846.0	100	Alabama	Shelby	NaN	Shelby
4	58669.0	33928.0	100	Alabama	Montgomery	NaN	Montgomery
5	18409.0	72780.0	100	Alabama	Baldwin	NaN	Baldwin
6	31746.0	47701.0	100	Alabama	Tuscaloosa	NaN	Tuscaloosa
7	20987.0	34321.0	100	Alabama	Lee	NaN	Lee
8	11216.0	37392.0	100	Alabama	Morgan	NaN	Morgan
9	13197.0	32803.0	100	Alabama	Calhoun	NaN	Calhoun
10	10350.0	32132.0	100	Alabama	Etowah	NaN	Etowah
11	10547.0	30567.0	100	Alabama	Houston	NaN	Houston
12	9340.0	28824.0	100	Alabama	Limestone	NaN	Limestone
13	9877.0	27735.0	100	Alabama	Lauderdale	NaN	Lauderdale
14	5550.0	31579.0	100	Alabama	St. Clair	NaN	St. Clair
15	3730.0	32734.0	100	Alabama	Cullman	NaN	Cullman
16	8436.0	27619.0	100	Alabama	Elmore	NaN	Elmore
17	4913.0	29217.0	100	Alabama	Marshall	NaN	Marshall
18	12108.0	20596.0	100	Alabama	Talladega	NaN	Talladega
19	4486.0	24208.0	100	Alabama	Walker	NaN	Walker
20	3682.0	21779.0	100	Alabama	DeKalb	NaN	DeKalb
21	2150.0	22808.0	100	Alabama	Blount	NaN	Blount
22	5908.0	18110.0	100	Alabama	Autauga	NaN	Autauga
23	7296.0	16718.0	100	Alabama	Colbert	NaN	Colbert
24	3663.0	16643.0	100	Alabama	Jackson	NaN	Jackson
25	4194.0	15825.0	100	Alabama	Coffee	NaN	Coffee
26	9577.0	9210.0	100	Alabama	Russell	NaN	Russell
27	12826.0	5784.0	100	Alabama	Dallas	NaN	Dallas
28	4408.0	13798.0	100	Alabama	Dale	NaN	Dale
29	5271.0	12967.0	100	Alabama	Tallapoosa	NaN	Tallapoosa
...	...	...	...	...	...	...	...
4554	2171.0	4564.0	100	Wisconsin	Rusk	NaN	Rusk
4555	2531.0	4049.0	100	Wisconsin	Buffalo	NaN	Buffalo
4556	1583.0	2787.0	100	Wisconsin	Forest	NaN	Forest
4557	1345.0	2228.0	100	Wisconsin	Pepin	NaN	Pepin

	Clinton	Trump	Rpt	State	Vote by county	Vote by town	Place
4558	1273.0	2090.0	100	Wisconsin	Iron	NaN	Iron
4559	666.0	1897.0	100	Wisconsin	Florence	NaN	Florence
4560	1003.0	269.0	100	Wisconsin	Menominee	NaN	Menominee
4561	11572.0	24844.0	100	Wyoming	Laramie	NaN	Laramie
4562	6573.0	23523.0	100	Wyoming	Natrona	NaN	Natrona
4563	1324.0	15778.0	100	Wyoming	Campbell	NaN	Campbell
4564	3233.0	12153.0	100	Wyoming	Sweetwater	NaN	Sweetwater
4565	4200.0	11167.0	100	Wyoming	Fremont	NaN	Fremont
4566	6888.0	7601.0	100	Wyoming	Albany	NaN	Albany
4567	2535.0	11115.0	100	Wyoming	Park	NaN	Park
4568	2926.0	10266.0	100	Wyoming	Sheridan	NaN	Sheridan
4569	7313.0	3920.0	100	Wyoming	Teton	NaN	Teton
4570	1105.0	6779.0	100	Wyoming	Lincoln	NaN	Lincoln
4571	1202.0	6154.0	100	Wyoming	Uinta	NaN	Uinta
4572	668.0	5520.0	100	Wyoming	Converse	NaN	Converse
4573	1279.0	4409.0	100	Wyoming	Carbon	NaN	Carbon
4574	924.0	4418.0	100	Wyoming	Goshen	NaN	Goshen
4575	594.0	4067.0	100	Wyoming	Big Horn	NaN	Big Horn
4576	719.0	3437.0	100	Wyoming	Platte	NaN	Platte
4577	638.0	3477.0	100	Wyoming	Johnson	NaN	Johnson
4578	644.0	3409.0	100	Wyoming	Sublette	NaN	Sublette
4579	271.0	3347.0	100	Wyoming	Crook	NaN	Crook
4580	532.0	2911.0	100	Wyoming	Washakie	NaN	Washakie
4581	294.0	2898.0	100	Wyoming	Weston	NaN	Weston
4582	400.0	1939.0	100	Wyoming	Hot Springs	NaN	Hot Springs
4583	115.0	1116.0	100	Wyoming	Niobrara	NaN	Niobrara

4584 rows × 7 columns

It can be seen the data are sorted but the total vote counts does not appear for each states.

The first part of this analysis will be to organize and count the vote for each states

In [22]:
```
# Getting the states participating into the vote from the file
# The third represents the states but that column has duplicates states' name
label = dataset.iloc[:, 3].values
states = []

for i in range(len(label)):
    if i < len(label) - 2:
        # Getting one state's name and avoiding duplicates
        if label[i] != label[i+1]:
            states.append(label[i])
# Append the last state into the list
states.append(label[len(label)-1])

# Show the states
print(states)
```

```
['Alabama', 'Arizona', 'Arkansas', 'California', 'Colorado', 'Connecticut',
'Delaware', 'Florida', 'Georgia', 'Hawaii', 'Idaho', 'Illinois', 'Indiana',
'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine', 'Maryland', 'Massachusett
s', 'Michigan', 'Minnesota', 'Mississippi', 'Missouri', 'Montana', 'Nebrask
a', 'Nevada', 'New Hampshire', 'New Jersey', 'New Mexico', 'New York', 'North
Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon', 'Pennsylvania', 'Rho
de Island', 'South Carolina', 'South Dakota', 'Tennessee', 'Texas', 'Utah',
'Vermont', 'Virginia', 'Washington', 'West Virginia', 'Wisconsin', 'Wyoming']
```

In [32]:
```
votes = []
votes.append(states)
print(votes)
```

```
[['Alabama', 'Arizona', 'Arkansas', 'California', 'Colorado', 'Connecticut',
'Delaware', 'Florida', 'Georgia', 'Hawaii', 'Idaho', 'Illinois', 'Indiana',
'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine', 'Maryland', 'Massachusett
s', 'Michigan', 'Minnesota', 'Mississippi', 'Missouri', 'Montana', 'Nebrask
a', 'Nevada', 'New Hampshire', 'New Jersey', 'New Mexico', 'New York', 'North
Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon', 'Pennsylvania', 'Rho
de Island', 'South Carolina', 'South Dakota', 'Tennessee', 'Texas', 'Utah',
'Vermont', 'Virginia', 'Washington', 'West Virginia', 'Wisconsin', 'Wyomin
g']]
```

In [33]:
```
# if 0 is passed then it will count the votes for Clinton and if it will count
the votes for Trump
def countVotes(canditate):

    # The total number of states that appear in the csv file is 48. Therefore
 a 2 dimensional array will be enough to store all the votes
    votePerState = []
    votePerState.append(states)

    # Get the votes column
    vote = dataset.iloc[:,canditate].values
    j = 0

    # Create an empty list to save the vote count for each states
    voteCount = [0]*49

    #Count the vote per state
    for i in range(len(label) - 1):
        if i < len(label) - 2:
            if label[i] == label[i+1]:
                voteCount[j] = voteCount[j] + vote[i] + vote[i+1]
            else:
                j = j + 1

    votePerState.append(voteCount)
    return votePerState

votes = countVotes(0)
print(votes)
```

```
[['Alabama', 'Arizona', 'Arkansas', 'California', 'Colorado', 'Connecticut',
'Delaware', 'Florida', 'Georgia', 'Hawaii', 'Idaho', 'Illinois', 'Indiana',
'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine', 'Maryland', 'Massachusett
s', 'Michigan', 'Minnesota', 'Mississippi', 'Missouri', 'Montana', 'Nebrask
a', 'Nevada', 'New Hampshire', 'New Jersey', 'New Mexico', 'New York', 'North
Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon', 'Pennsylvania', 'Rho
de Island', 'South Carolina', 'South Dakota', 'Tennessee', 'Texas', 'Utah',
'Vermont', 'Virginia', 'Washington', 'West Virginia', 'Wisconsin', 'Wyomin
g'], [1281057.0, 1251296.0, 667445.0, 9375852.0, 2310118.0, 1613486.0, 27491
0.0, 8347833.0, 3392180.0, 341564.0, 303634.0, 4426040.0, 1850544.0, 1181344.
0, 703169.0, 1066622.0, 1455030.0, 681241.0, 2682657.0, 3712757.0, 4017197.0,
2303392.0, 858685.0, 1828717.0, 327895.0, 442493.0, 674373.0, 671548.0, 37385
19.0, 619332.0, 7691244.0, 4025090.0, 155719.0, 4248696.0, 726691.0, 1597355.
0, 5128399.0, 412033.0, 1622878.0, 203978.0, 1527003.0, 7029157.0, 327973.0,
341703.0, 3478834.0, 1952676.0, 346546.0, 2474431.0, 99696.0]]
```

In [34]:
```
# indexes will be generates to plots the states names on the x-axis
index1 = np.arange(len(states))
ind = np.arange(len(votes[1]))
index1
```

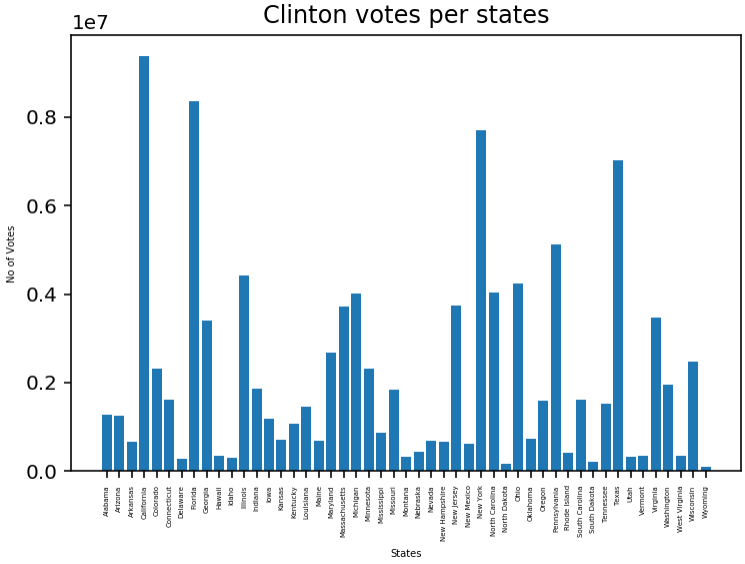
Out[34]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48])
```

In [35]:
```
# Plot function to have an overall view of the results
def barPlot(x,y,candidate,candidateName):
    plt.bar(x,y)
    plt.xlabel('States',fontsize=5)
    plt.ylabel('No of Votes',fontsize=5)
    plt.xticks(index1,states, fontsize = 3.5, rotation=90)
    plt.title(candidateName + ' votes per states')
    plt.show()
```

Clinton votes per states

In [36]:
```
barPlot(index1,votes[1],0,'Clinton')
```

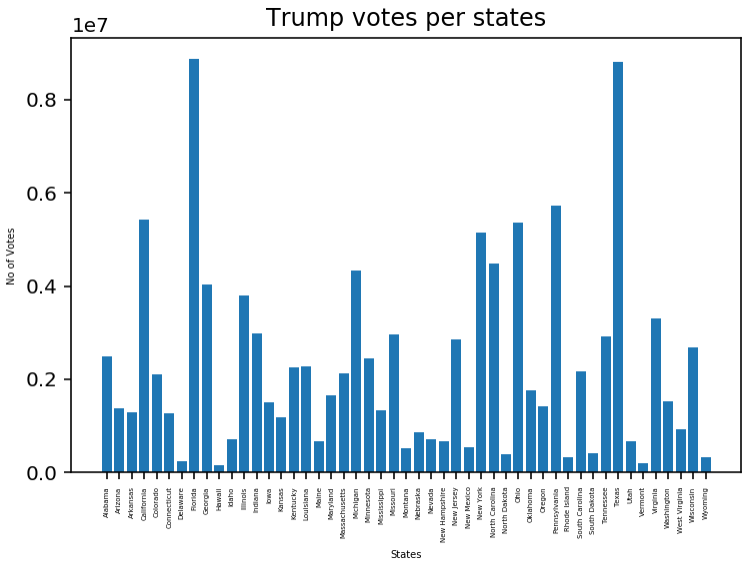
Out[36]:



Trump votes per states

In [37]:
```
votes2 = countVotes(1)
barPlot(index1,votes2[1],1,'Trump')
```

Out[37]:



As it can be seen the distribution of the votes are similar that might indicate that there is may be not a huge difference in the results

In [0]:
```

```