Party Classifier

May 29, 2020

1 Party Classification by Tweet

In this notebook I will build a model that classifies tweets as either coming from a Democrat or a Republican politcian. To train my model I will use a collection of the latest 200 tweets (as of May 17th, 2018) from every member of the House of Representatives. This data set can be found at https://www.kaggle.com/kapastor/democratvsrepublicantweets#ExtractedTweets.csv

[79]:
```
%matplotlib inline
import string
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
matplotlib.rc('xtick', labelsize=14)
matplotlib.rc('ytick', labelsize=14)
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

from sklearn import svm
from sklearn.svm import SVC
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
import re
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import chi2, SelectKBest
from sklearn.model_selection import GridSearchCV
```

The collection of tweets is stored under the variable 'data', the corresponding party label will stored stored under 'targets', we see that the data set has 86,460 samples

[3]:
```
data=pd.read_csv('ExtractedTweets.csv')['Tweet']
targets=pd.read_csv('ExtractedTweets.csv')[['Party']]
data.shape
```

[3]:
```
(86460,)
```

First, I do some preprocessing, the following for loop removes all punctuation, digits, and URLs from the tweet data set

[128]:
```
for i in range(len(data)):
    data[i]=data[i].replace("’","")
    data[i]=data[i].replace("'","")
    data[i]=re.sub(r"http\S+", "", data[i])
    data[i]=' '.join(s for s in data[i].split() if not any(c.isdigit() for c in␣
 ↪s))
    data[i]=data[i].translate(str.maketrans('', '', string.punctuation))
```

Next, I use the CountVectorizer utility to turn the tweets into a bag of words model, this will convert each tweet into a vector over the space of possible words. The value of a given dimension of the vector corresponds to the number of times that word appears in the tweet.

[130]:
```
count_vect = CountVectorizer(ngram_range=(1,2))
X_train_counts = count_vect.fit_transform(data)
X_train_counts.shape
```

[130]:
```
(86460, 523474)
```

We can see from the output of X_train_counts.shape that there are a total of 523,474 different unigrams and bigrams (groups of one or two words) in the corpus

Next I use the utility TfidfTransformer, this will scale the count vectors to give more weight to unique words in the data set and give less weight to very common words e.g.(the, as, we etc.)

[131]:
```
tfidf = TfidfVectorizer(ngram_range=(1,2))
X_train_tfidf = tfidf.fit_transform(data)
```

Next, I shuffle the order of the data so that the initial ordering does not bias the classifier.

[132]:
```
X,y=shuffle(X_train_tfidf,np.ravel(targets.values))
```

Finally I define a feature selector, SelectKBest this will only select the top k features to use in the classifier. Where the top features are measured by the chi2 statistic

[133]:
```
sel=SelectKBest(score_func=chi2)
```

Now the data is ready to be fed into a classification model, I will try out several different types of classifiers.

In the following cell I fit the data to a multinomial Naive Bayes model and estimate the accuracy using 5-fold cross validation. The optimal value of alpha is found using grid search

[146]:
```
clf = MultinomialNB()

pipeline=Pipeline([('sel',sel),('clf',clf)])
parameters = {'sel__k': [1000,10000,100000],
```

```
            'clf__alpha':[.01,.1,1]}
model = GridSearchCV(pipeline,parameters, cv=5, iid=False,␣
 ↪n_jobs=-1,refit=True,verbose=5,scoring='accuracy')
model.fit(X,y)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    4.8s
[Parallel(n_jobs=-1)]: Done  40 out of  45 | elapsed:   24.5s remaining:    3.1s
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed:   27.9s finished
```

[146]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('sel', SelectKBest(k=10, score_func=<function chi2 at
0x124709c80>)), ('clf', MultinomialNB(alpha=1.0, class_prior=None,
fit_prior=True))]),
       fit_params=None, iid=False, n_jobs=-1,
       param_grid={'sel__k': [1000, 10000, 100000], 'clf__alpha': [0.01, 0.1,
1]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=5)
```

[147]:
```
model.best_params_
```

[147]:
```
{'clf__alpha': 0.1, 'sel__k': 100000}
```

[148]:
```
model.best_score_
```

[148]:
```
0.8022206220333707
```

We see that with the optimal parameters of alpha= 0.1 and k= 100,000 , the NB algorithm has about an 80% success rate. Next I try a Logistic Regression model:

[143]:
```
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression(solver='liblinear')
pipeline=Pipeline([('sel',sel),('clf',clf)])
parameters = {'sel__k': [1000,10000,100000,500000],
            'clf__C':[.1,1,10,100]}
model = GridSearchCV(pipeline,parameters, cv=5, iid=False,␣
 ↪n_jobs=-1,refit=True,verbose=5,scoring='accuracy')
model.fit(X,y)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    4.6s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   42.1s
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:  1.1min finished
```

[143]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('sel', SelectKBest(k=10, score_func=<function chi2 at
0x124709c80>)), ('clf', LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False))]),
       fit_params=None, iid=False, n_jobs=-1,
       param_grid={'sel__k': [1000, 10000, 100000, 500000], 'clf__C': [0.1, 1,
10, 100]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=5)
```

[144]:
```
model.best_params_
```

[144]:
```
{'clf__C': 10, 'sel__k': 100000}
```

[145]:
```
model.best_score_
```

[145]:
```
0.8033309406727456
```

Again, we see about an 80% classification accuracy. Next I try a Support Vector Machine with a linear decision boundary. (Full grid search code omitted)

[ ]:
```
from sklearn.svm import LinearSVC
clf = svm.LinearSVC(C=.8, loss='squared_hinge',max_iter=5000)
cross_val_score(clf, X, y, cv=5).mean()
```

Finally, I try a Decision Tree and a Random Forest Classifier (Full grid search code omitted):

[11]:
```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth=100)
cross_val_score(clf, X,y, cv=5).mean()
```

[11]:
```
0.6313788197214114
```

[12]:
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100,max_depth=10)
cross_val_score(clf, X,y, cv=5).mean()
```

[12]:
```
0.6195581801878173
```

We see that the Naive Bayes, Logistic Regression, and Support Vector Machine algorithms all have an accuracy around 80%, while the Decision Tree and Random Forest classifiers are only 62-63% accurate

As of now, my model seems to maxing out at 80% accuracy, how can this be improved upon? What might be causing our 20% error rate? One reason may be the quality of our data, one can see by

examining this data set that many tweets are cut off significantly, for example let's look at the first tweet:

[58]:
```
data[0]
```

[58]:
```
'Today, Senate Dems vote to #SaveTheInternet. Proud to support similar
#NetNeutrality legislation here in the House…'
```

A Google search lead me to the full tweet:

"Today, Senate Dems vote to #SaveTheInternet. Proud to support similar #NetNeutrality legislation here in the House to protect small businesses & innovators in our district. The fight for a free and open internet continues! #Sayfie"

One can see we are missing lots of valuable information by cutting off the tweets like this, especially from the hashtags that are often at the end of the tweet.

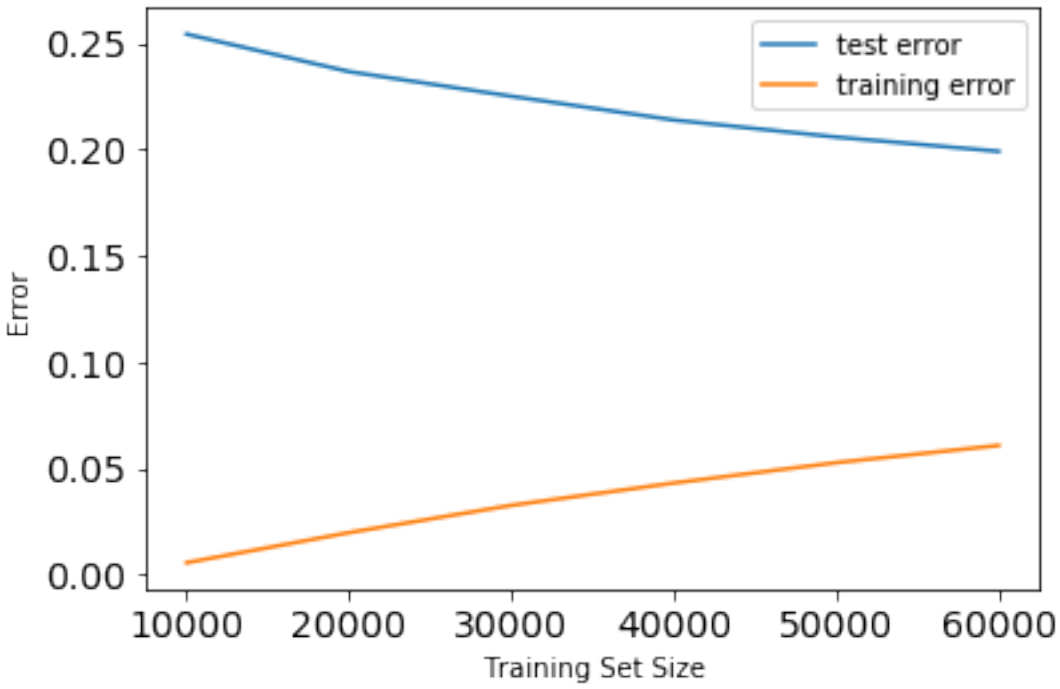
Another cause of error may be an insufficient quanitity of data. Would one expect the model to perform better if fed a larger data set? Below I plot the training and test error for the Naive Bayes model as a function of training set size.

[87]:
```
data_size=[10000,20000,30000,40000,50000,60000]
test_error=[]
train_error=[]
clf = MultinomialNB(alpha=0.1)
sel=SelectKBest(score_func=chi2,k=100000)
pipeline=Pipeline([('sel',sel),('clf',clf)])



X_train, X_test, y_train, y_test = train_test_split( X_train_tfidf, np.
 ↪ravel(targets.values), test_size=0.3,random_state=42)
for i in data_size:
    pipeline.fit(X_train[0:i,:],y_train[0:i])
    train_pred=pipeline.predict(X_train[0:i,:])
    y_prediction = pipeline.predict(X_test)
    test_error.append(1-accuracy_score(y_true = y_test, y_pred = y_prediction))
    train_error.append(1-accuracy_score(y_true = y_train[0:i], y_pred =␣
 ↪train_pred))
plt.plot(data_size,test_error,label='test error')
plt.plot(data_size,train_error,label='training error')
plt.legend()
plt.xlabel('Training Set Size')
plt.ylabel('Error')
```

[87]:
```
Text(0, 0.5, 'Error')
```



We see that as I give the model more training data, its test error goes down and its training error goes up. This is a sign that the model is not underfitting the data and that if given more data it would continue to improve.

Finally, I examine my classifier to see which words most strongly influenced the model to classify a tweet as either Democrat or Republican. This is most straightforward using Logistic Regression.

The list of word coefficients is given by applying method, '.coef_' to our classifier

[149]:
```
clf=LogisticRegression(solver='liblinear',C=10)
clf.fit(X,y)
```

[149]:
```
LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

Next we order the coefficients, a higher negative value indicates a more Democrat correlated word and a higher positive value indicates a more Republican one. We find the ten lowest coefficents and look at their corresponding words

[150]:
```
most=np.argsort(clf.coef_)[0,0:11]
[count_vect.get_feature_names()[i] for i in most]
```

[150]:
```
['goptaxscam',
 'trump',
 'netneutrality',
 'trumps',
 'energycommerce',
 'repdarrensoto',
 'officialcbc',
 'pruitt',
 'republicans',
 'dreamers',
 'housedemocrats']
```

The viral hashtag #goptaxscam referring to the Republican tax bill is (obviously) a strong indicator of a Democratic tweet. In addition, one sees names of democratic congressmen like 'repdarrensoto' and progressive issues like 'netneutrality'.

Next I look at the most Republican correlated words:

[151]:
```
most=np.argsort(clf.coef_)[0,-11:-1]
for i in reversed([count_vect.get_feature_names()[i] for i in most]):
    print(i)
```

```
rt housegop
housecommerce
taxcutsandjobsact
chairman
rt speakerryan
foxnews
schumershutdown
obamacare
rt realdonaldtrump
repmarkwalker
```

We see that the more positive description of the Republican tax bill, 'taxcutsandjobsact' is strongly correlated with Republicans, as well as names of Republican congressmen such as 'speakerryan'.