MasterCode

January 24, 2019

1 Project 1

2 Data Mining

2.1 ---

2.2 Question 1

Collect training data from 20 Newsgroups data set, and plot a histogram of each label's occurances

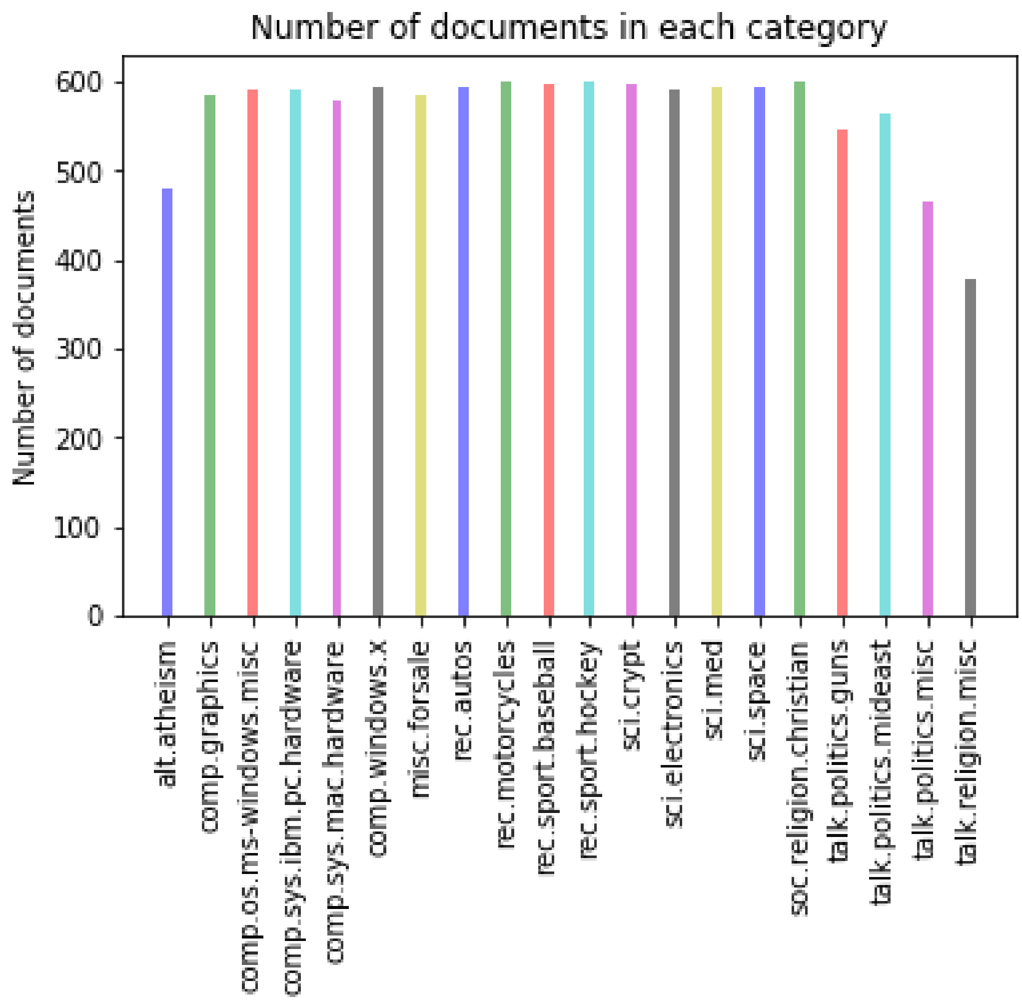
```
In [1]: #Code from Sarat for Q1
        from sklearn.datasets import fetch_20newsgroups
        from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
        from matplotlib import pylab as plt
        import matplotlib.colors as pltc
        from itertools import cycle
        import numpy as np
        np.random.seed(42)
        import random
        random.seed(42)
```

```
In [2]: twenty_train_dataset = fetch_20newsgroups(subset="train",
                                                  shuffle=True,
                                                  random_state=42)
        twenty_test_dataset  = fetch_20newsgroups(subset="test",
                                                 shuffle=True,
                                                 random_state=42)
        # bad choice of variable name.
        cat_labels = np.unique(twenty_train_dataset.target)
        print('Category labels present in the data set are: ', cat_labels)
        num_classes = len(np.unique(twenty_train_dataset.target))
        [freq_classes, _] = np.histogram(twenty_train_dataset.target,
                                         bins=np.arange(0,num_classes+1))
```

```
Category labels present in the data set are:  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```
In [3]: cycl = cycle('bgrcmky')
        for i in range(len(cat_labels)):
```

```
    plt.bar([cat_labels[i]], [freq_classes[i]], width=0.25,
            align='center', color=next(cycl),
            alpha=0.5, label=twenty_train_dataset.target_names[i])
plt.xticks(cat_labels, twenty_train_dataset.target_names, rotation=90)
# plt.legend()
plt.ylabel('Number of documents')
plt.subplots_adjust(bottom= 0.22, top = 0.95)
plt.title("Number of documents in each category")
plt.show()
```



Code given to us to maintain consistency. Loads Train and Test datasets.

The first 4 categories are computer related (we will call this class 0) and the second 4 are recreation related (we will call this class 1).

2.3 Question 2

Clean the data. Remove stopwords, perform lemmatization. Use the CountVectorizer to build the doc-term frequency matrix. Use the TfidfTransformer to build the TF-IDF matrix!

```
In [4]: #Q2: Extract features w/ particular specifications
        #Taken from Shannon
        import nltk
        from nltk import pos_tag
        from pickle import dump

        comp_categories = [ 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardwa
                           'comp.sys.mac.hardware']
        rec_categories = ['rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hock
        train_dataset = fetch_20newsgroups(subset='train', categories=comp_categories+rec_categ
                                          shuffle=True, random_state=42,)
        test_dataset = fetch_20newsgroups(subset='test', categories=comp_categories+rec_catego
                                         shuffle=True, random_state=42,)
        counts = []

        wnl = nltk.wordnet.WordNetLemmatizer()
        anlyzer = CountVectorizer().build_analyzer()

        def penn2morphy(penntag):
            """ Converts Penn Treebank tags to WordNet. """
            morphy_tag = {'NN':'n', 'JJ':'a',
                          'VB':'v', 'RB':'r'}
            try:
                return morphy_tag[penntag[:2]]
            except:
                return 'n'

        def lemmatize_sent(list_word):
            # Text input is string, returns array of lowercased strings(words).
            return [wnl.lemmatize(word.lower(), pos=penn2morphy(tag))
                    for word, tag in pos_tag(list_word)]

        #NOTE: The following code was developed to remove all numbers (including floats like 6
        #      as isdigit doesn't work on floats. However, the floats it removed were:
        #      ['35002_4401', '5e8', '5e9', '6e1', '9_6', 'inf', 'infinity']
        #      So we decided not to use it
        #def is_number(s):
        #    try:
        #        float(s)
        #        return True
        #    except:
        #        return False

        def rmv_nums(doc):
```

```
    #gets rid of numbers including floats
    #does lemmatization with nltk.wordnet.WordNetLemmatizer and pos_tag
    return (word for word in lemmatize_sent(anlyzer(doc))
            if not word.isdigit())


#CountVectorizer returns a callable that handles preprocessing and tokenization
#Use the english stopwords of the CountVectorizer
vectorizer=CountVectorizer(analyzer=rmv_nums,min_df=3,stop_words='english')

#do feature extraction (train):
X_train_counts=vectorizer.fit_transform(train_dataset.data) #get matrix of doc-term co
print('Size of training data after lemmatization but before TF-IDF: ', X_train_counts.s
X_test_counts=vectorizer.transform(test_dataset.data)
print('Size of testing data after lemmatization but before TF-IDF:  ', X_test_counts.sl

from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
print('Shape of train TF-IDF matrix: ',X_train_tfidf.shape)
X_test_tfidf = tfidf_transformer.transform(X_test_counts)
print('Shape of test TF-IDF matrix:  ',X_test_tfidf.shape)
```

```
Size of training data after lemmatization but before TF-IDF:  (4732, 16600)
Size of testing data after lemmatization but before TF-IDF:   (3150, 16600)
Shape of train TF-IDF matrix:  (4732, 16600)
Shape of test TF-IDF matrix:   (3150, 16600)
```

As we can see, the shape of the category count matrices for the train and test data set are 4732x16600 and 3150x16600 respectively. In addition, the shape doesn't change once the TF-IDF transform is applied.

2.4 Question 3: Apply Dimensionality Reduction

Do dimensionality reduction with LSI. In this part, I create the SVD object and transform the test date into a reduced version of iself.

```
In [5]: #Q3
        #LSA
        from sklearn.decomposition import TruncatedSVD

        svd = TruncatedSVD(n_components=50, random_state=42)
        X_train_LSA = svd.fit_transform(X_train_tfidf)
        X_test_LSA = svd.transform(X_test_tfidf)
        print('SVD reduced training set shape: ', X_train_LSA.shape)
        print('SVD reduced testing set shape:  ', X_test_LSA.shape)

        V = svd.components_
```

```
SVD reduced training set shape:  (4732, 50)
SVD reduced testing set shape:   (3150, 50)
```

```
In [6]: LSA_train_error=np.sum(np.array(X_train_tfidf - X_train_LSA.dot(V))**2)
        LSA_test_error=np.sum(np.array(X_test_tfidf - X_test_LSA.dot(V))**2)

        print('Error from LSA training set: ', LSA_train_error)
        print('Error from LSA testing set:  ', LSA_test_error)
```

```
Error from LSA training set:  3895.4186796933272
Error from LSA testing set:   2676.4912327221027
```

Do dimensionality reduction with NMF. Make the NMF object, witch find a W and H, and then uses H to transform the test data too.

```
In [7]: #Q3 cont
        #NMF
        from sklearn.decomposition import NMF
        nmf = NMF(n_components=50, init='random', random_state=42)

        W_train = nmf.fit_transform(X_train_tfidf) #Find our W and H from the train data
        W_test  = nmf.transform(X_test_tfidf) #apply the found H and do the minimization

        H = nmf.components_

        print('NMF reduced (W) training set shape: ', W_train.shape)
        print('NMF reduced (W) testing set shape:  ', W_test.shape)
        print('NMF reduced (H) set shape:          ', H.shape)
```

```
NMF reduced (W) training set shape:  (4732, 50)
NMF reduced (W) testing set shape:   (3150, 50)
NMF reduced (H) set shape:           (50, 16600)
```

```
In [8]: #Get NMF error
        NMF_train_error = np.sum(np.array(X_train_tfidf - W_train.dot(H))**2)
        NMF_test_error = np.sum(np.array(X_test_tfidf - W_test.dot(H))**2)

        print('Error from NMF training set: ', NMF_train_error)
        print('Error from NMF testing set:  ', NMF_test_error)
```

```
Error from NMF training set:  3940.342513932874
Error from NMF testing set:   2689.0690267386617
```

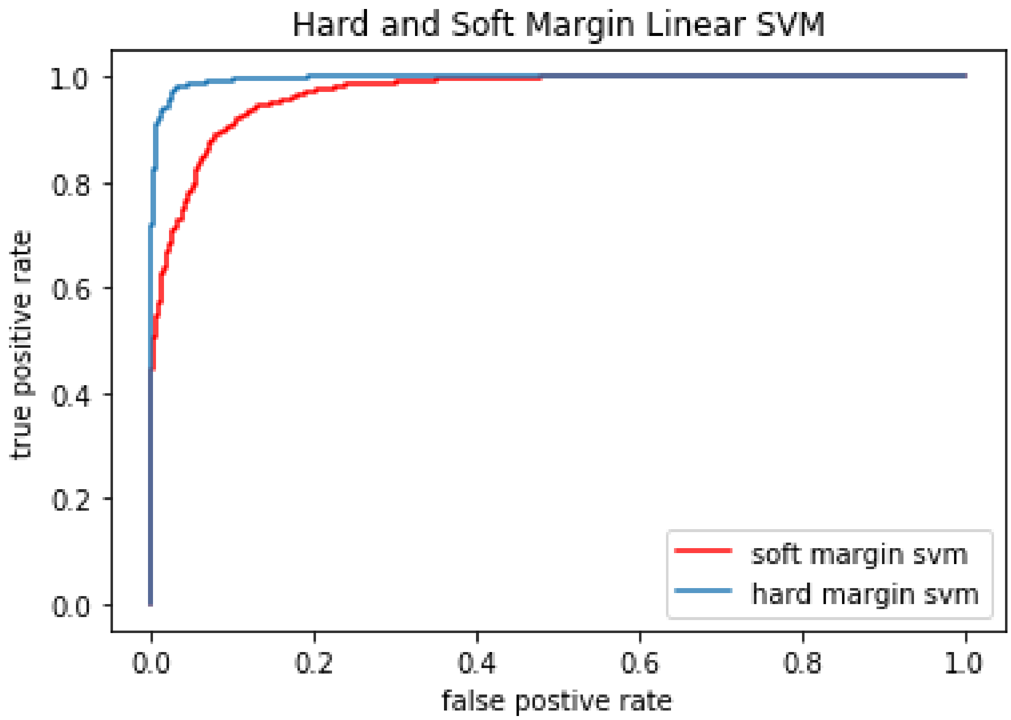
(!!!!!More Info!!!!) LSA is nothing but PCA which gives the optimal projection in terms of Squared error. Read about NMF

2.5 Question 4: Classification Algorithms

In [9]:
```
#Question 4, code from Sarat
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import auc

y_train = train_dataset.target > 3
y_train = y_train.astype(int)
y_test = test_dataset.target > 3
y_test = y_test.astype(int)

#hard
clf_hard = LinearSVC(C=1000, random_state=42, max_iter=100000).fit(X_train_LSA, y_trai
y_pred_hard = clf_hard.predict(X_test_LSA)
scores_hard = clf_hard.decision_function(X_test_LSA)

#soft
clf_soft = LinearSVC(C=0.0001, random_state=42, max_iter=100000).fit(X_train_LSA, y_tra
y_pred_soft = clf_soft.predict(X_test_LSA)
scores_soft = clf_soft.decision_function(X_test_LSA)
```

ROC Curve

In [10]:
```
# roc curve:
fpr_hard, tpr_hard, thresholds_hard = roc_curve(y_test, scores_hard)
fpr_soft, tpr_soft, thresholds_soft = roc_curve(y_test, scores_soft)

# plot roc curves:

plt.plot(fpr_soft, tpr_soft, 'r', label='soft margin svm')
plt.plot(fpr_hard, tpr_hard, label='hard margin svm')
plt.legend()
plt.title('Hard and Soft Margin Linear SVM')
plt.xlabel('false postive rate')
plt.ylabel('true positive rate')
plt.show()
```



As we want the area under the ROC curve to be high, Hard margin SVM is a better choice in this problem.

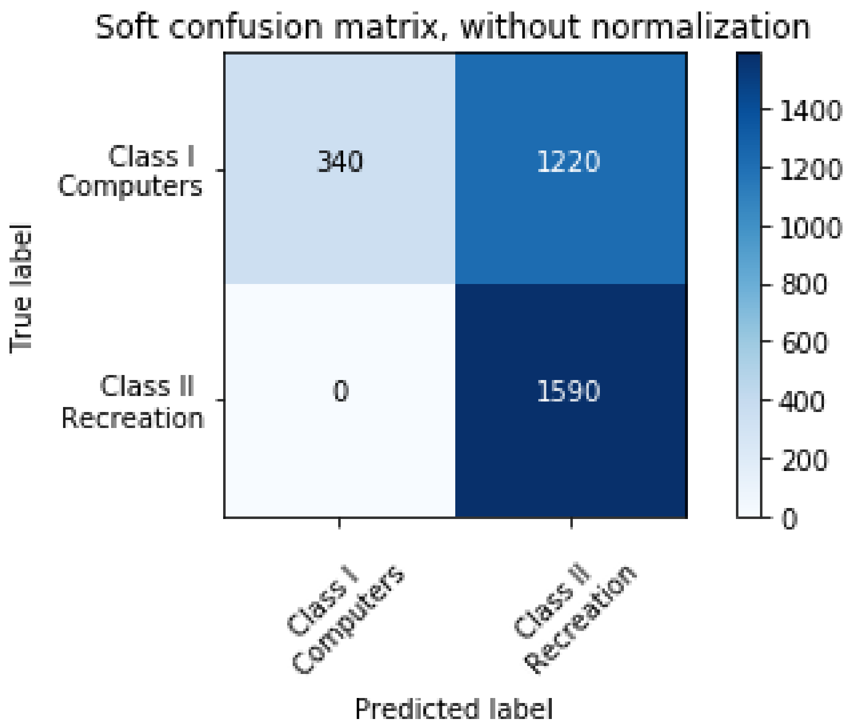
Confusion Matrix

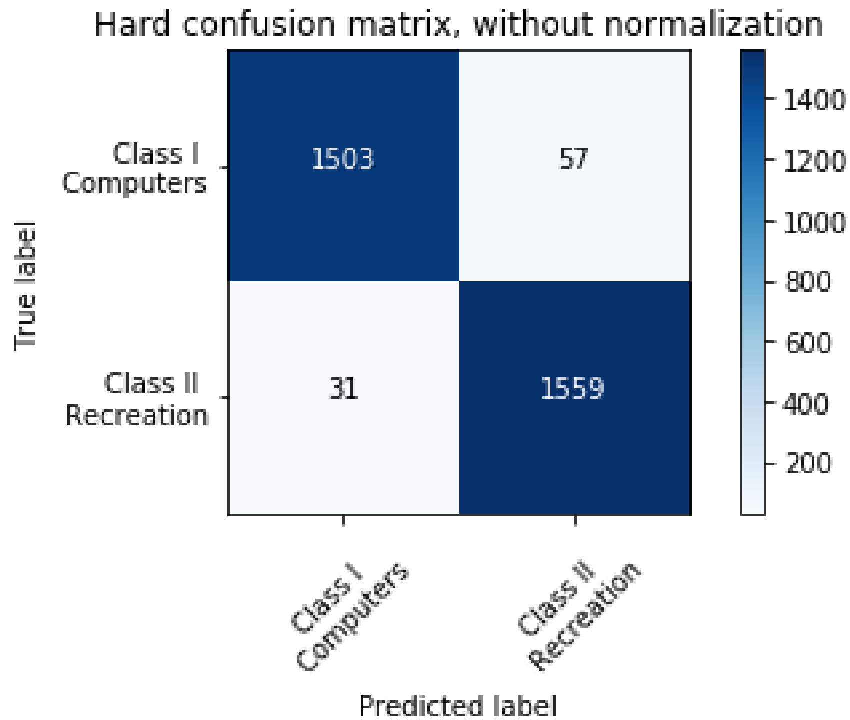
In [11]:
```
confusion_hard = confusion_matrix(y_test, y_pred_hard)
confusion_soft = confusion_matrix(y_test, y_pred_soft)
```

In [12]:
```
#confusion matrix plot
import itertools

def plot_confusion_matrix(cm, title, cmap = plt.cm.Blues):
    ctgrs = ['Class I \n Computers','Class II \n Recreation']
    plt.figure()
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title + ', without normalization')
    plt.colorbar()
    tick_marks = np.arange(len(ctgrs))
    plt.xticks(tick_marks, ctgrs, rotation=45)
    plt.yticks(tick_marks, ctgrs)

    fmt = 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
```

```
                         horizontalalignment="center",
                         color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

In [13]:
```
# Plot non-normalized confusion matrices
plot_confusion_matrix(confusion_soft, title='Soft confusion matrix')
plot_confusion_matrix(confusion_hard, title='Hard confusion matrix')
```





Accuracy, Recall, Precision and F1 Score

In [14]:
```
#hard
accuracy_hard  = accuracy_score(y_test, y_pred_hard)
recall_hard    = recall_score(y_test, y_pred_hard)
precision_hard = precision_score(y_test, y_pred_hard)
f1_score_hard  = f1_score(y_test, y_pred_hard)

#soft
accuracy_soft = accuracy_score(y_test, y_pred_soft)
recall_soft = recall_score(y_test, y_pred_soft)
precision_soft = precision_score(y_test, y_pred_soft)
f1_score_soft = f1_score(y_test, y_pred_soft)

print('Hard accuracy:  ', accuracy_hard,  '    Soft accuracy:  ', accuracy_soft)
print('Hard recall:    ', recall_hard,    '     Soft recall:    ', recall_soft)
print('Hard precision: ', precision_hard, '    Soft precision: ', precision_soft)
print('Hard F1 score:  ', f1_score_hard,  '    Soft F1 score:  ', f1_score_soft)
```

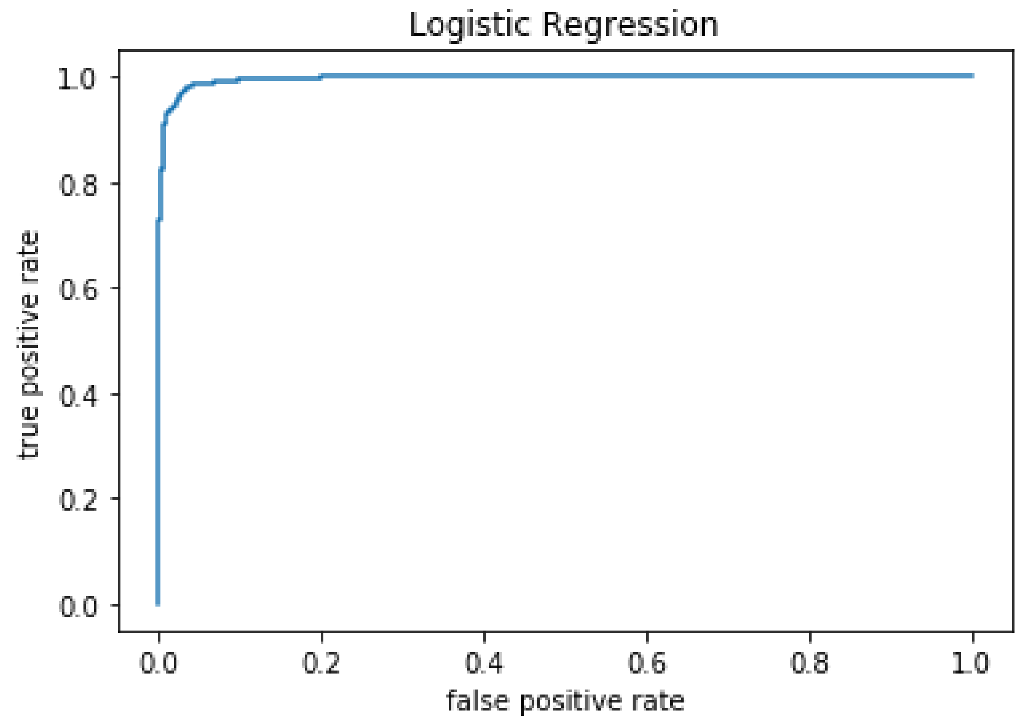
```
Hard accuracy:   0.9720634920634921     Soft accuracy:   0.6126984126984127
Hard recall:     0.980503144654088      Soft recall:     1.0
Hard precision:  0.9647277227722773     Soft precision:  0.5658362989323843
Hard F1 score:   0.9725514660012476     Soft F1 score:   0.7227272727272728
```

2.6 Question 5

In [15]:
```python
#Combined code from Eden and Sarat
from sklearn.linear_model import LogisticRegression

# C very high value to simulate no regularization
clf = LogisticRegression(C=1e10, random_state=42, solver='lbfgs',
                         multi_class='multinomial', max_iter=500)
clf.fit(X_train_LSA, y_train)
y_pred = clf.predict(X_test_LSA)
unrg_scores_prob = clf.predict_proba(X_test_LSA)[:, 1]


predicted = y_pred[0:10]
print('predicted: ', predicted)
print('actual:    ', y_train[0:10])
```

```
predicted:  [0 0 0 0 0 0 0 1 1 0]
actual:     [1 1 1 0 0 0 0 1 1 0]
```

In [16]:
```python
# roc curve:
fpr, tpr, thresholds = roc_curve(
    y_test, unrg_scores_prob)

# plot roc curves:
plt.plot(fpr, tpr)
plt.title('Logistic Regression')
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```




```
In [17]: #defining logitistic types
         unreg_confusion = confusion_matrix(y_test, y_pred)
         unreg_accuracy = accuracy_score(y_test, y_pred)
         unreg_recall = recall_score(y_test, y_pred)
         unreg_precision = precision_score(y_test, y_pred)
         unreg_f1_score = f1_score(y_test, y_pred)

         print('Accuracy logistic:  ', unreg_accuracy) #
         print('Recall logistic:    ', unreg_recall) #
         print('Precision logistic: ', unreg_precision) #
         print('F1 score logistic:  ', unreg_f1_score) #
```

```
Accuracy logistic:   0.9714285714285714
Recall logistic:     0.9792452830188679
Precision logistic:  0.9646840148698885
F1 score logistic:   0.9719101123595505
```

2.6.1 Classifiers with Regularization

L1 Regularization

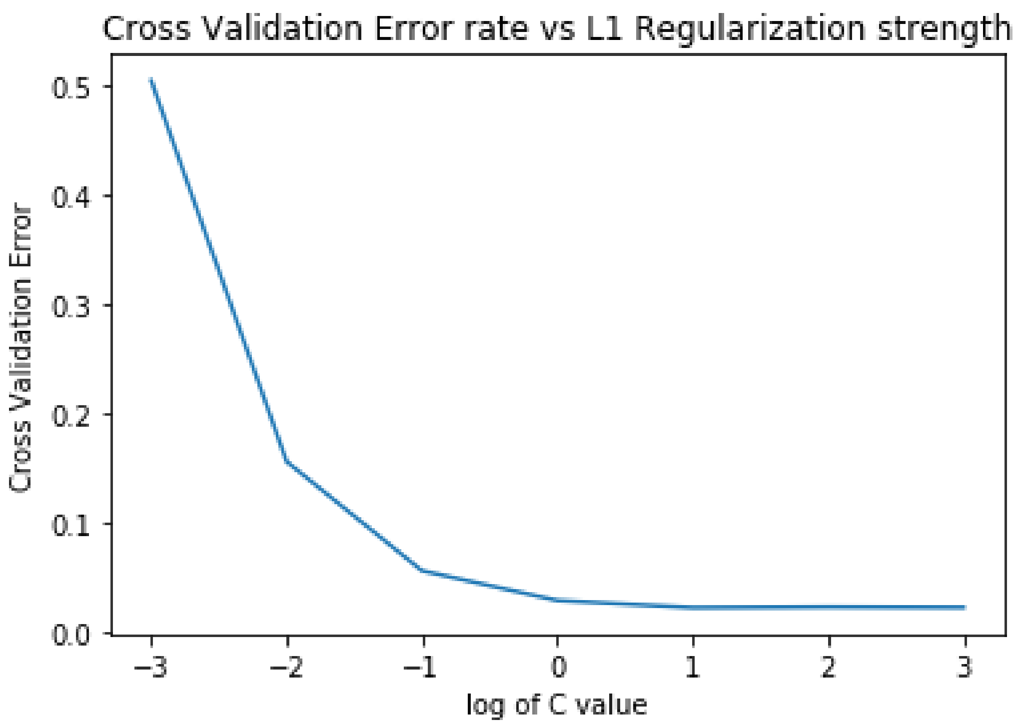
```
In [18]: from sklearn.linear_model import LogisticRegressionCV
         from sklearn.model_selection import cross_val_score
```

```
from math import log

C_pot = [0.001, 0.01, 0.1, 1, 10, 100, 1000] #potential regularization strengths for l
clf_l1 = LogisticRegressionCV(Cs=C_pot, cv=5, penalty='l1', refit=True,
                              solver='liblinear', random_state=42)
clf_l1.fit(X_train_LSA, y_train)
scores_l1_5fold = clf_l1.scores_
c_opt_l1 = clf_l1.C_[0]
k_opt_l1= log(c_opt_l1, 10)
inv_c_opt_l1 = c_opt_l1**-1 #Each of the values in Cs describes the inverse of regula
print('Optimal k-value for L1: ', k_opt_l1)
print('This corresponds to an optimal regularization strength for L1 (inverse of C=10
      inv_c_opt_l1)
```

```
Optimal k-value for L1:  1.0
This corresponds to an optimal regularization strength for L1 (inverse of C=10^k):  0.1
```

In [19]:
```
scores_l1 = []
for l in range(0,7):
    scores_l1.append(np.mean(scores_l1_5fold[1][:,l]))
k_values = range(-3,4)
plt.plot(np.asarray(k_values), 1-np.asarray(scores_l1))
plt.xlabel("log of C value")
plt.ylabel("Cross Validation Error")
plt.title("Cross Validation Error rate vs L1 Regularization strength")
```

Out[19]:
```
Text(0.5, 1.0, 'Cross Validation Error rate vs L1 Regularization strength')
```



We see in the plot above that regularization decreases the test error.

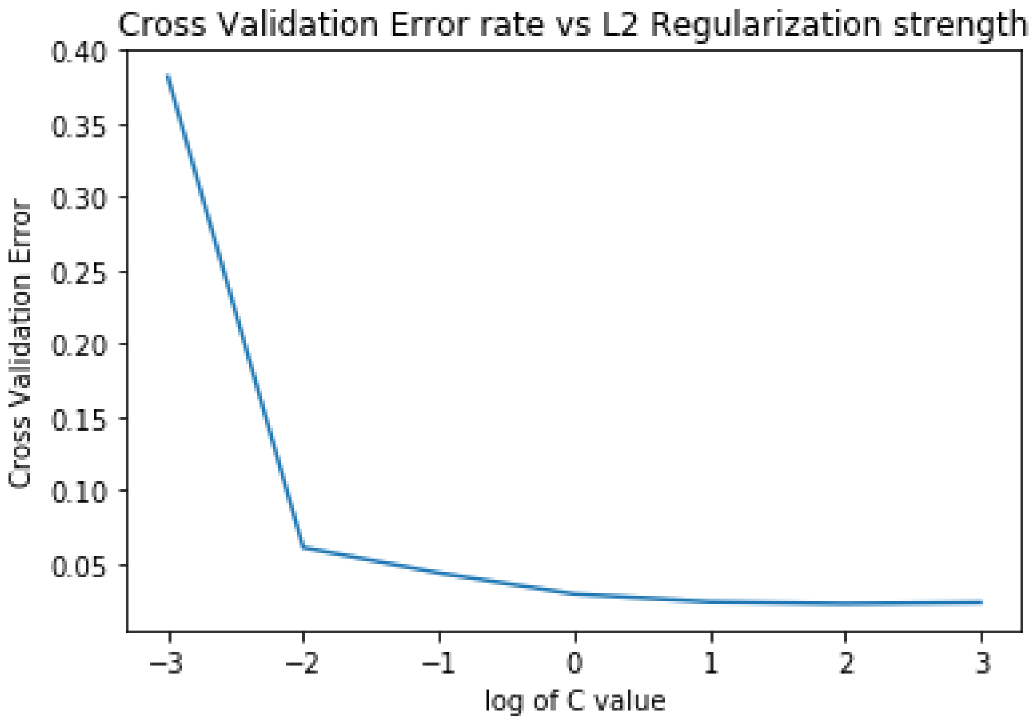
L2 Regularization

```
In [20]: clf_l2 = LogisticRegressionCV(Cs=C_pot, cv=5, penalty='l2', refit=True,
                                       solver='liblinear', random_state=42)
         clf_l2.fit(X_train_LSA, y_train)
         scores_l2_5fold = clf_l2.scores_
         c_opt_l2 = clf_l2.C_[0]
         k_opt_l2= log(c_opt_l2, 10)
         inv_c_opt_l2 = c_opt_l2**-1 #Each of the values in Cs describes the inverse of
                                     #regularization strength

         print('Optimal k-value for L2: ', inv_c_opt_l2)
         print('This corresponds to an optimal regularization strength for L2 (inverse of C=10
               inv_c_opt_l1)
```

```
Optimal k-value for L2:  0.01
This corresponds to an optimal regularization strength for L2 (inverse of C=10^k):  0.1
```

```
In [21]: scores_l2 = []
         for i in range(0,7):
             scores_l2.append(np.mean(scores_l2_5fold[1][:,i]))
         plt.plot(np.asarray(k_values), 1-np.asarray(scores_l2))
         plt.xlabel("log of C value")
         plt.ylabel("Cross Validation Error")
         plt.title("Cross Validation Error rate vs L2 Regularization strength")
```

```
Out[21]: Text(0.5, 1.0, 'Cross Validation Error rate vs L2 Regularization strength')
```



In [22]:
```
#Find the performance of best L1, best L2, No regularization test set performance

#L1:
clf_l1_opt    = LogisticRegression(penalty='l1', C=c_opt_l1, solver='liblinear',
                                            random_state=42)
clf_l1_opt.fit(X_train_LSA, y_train)
y_pred_l1_opt = clf_l1_opt.predict(X_test_LSA)
scores_l1_opt = clf_l1_opt.decision_function(X_test_LSA)

#L2
clf_l2_opt    = LogisticRegression(C=c_opt_l2, penalty='l2', solver='liblinear',
                                            random_state=42)
clf_l2_opt.fit(X_train_LSA, y_train)
y_pred_l2_opt = clf_l2_opt.predict(X_test_LSA)
scores_l2_opt = clf_l2_opt.decision_function(X_test_LSA)

#L1: accuracy, recall, precision, f1 score
accuracy_l1_opt  = accuracy_score(y_test, y_pred_l1_opt)
recall_l1_opt    = recall_score(y_test, y_pred_l1_opt)
precision_l1_opt = precision_score(y_test, y_pred_l1_opt)
f1_score_l1_opt  = f1_score(y_test, y_pred_l1_opt)

#L2: accuracy, recall, precision, f1 score
accuracy_l2_opt  = accuracy_score(y_test, y_pred_l2_opt)
```

```
recall_l2_opt    = recall_score(y_test, y_pred_l2_opt)
precision_l2_opt = precision_score(y_test, y_pred_l2_opt)
f1_score_l2_opt  = f1_score(y_test, y_pred_l2_opt)

print(' Accuracy score for unregularized: ', unreg_accuracy, '\n',
      '                          for l1: ', accuracy_l1_opt, '\n',
      '                          for l2: ', accuracy_l2_opt)
print('   Recall score for unregularized: ', unreg_recall, '\n',
      '                          for l1: ', recall_l1_opt, '\n',
      '                          for l2: ',  recall_l2_opt)
print('Precision score for unregularized: ', unreg_precision, '\n',
      '                          for l1: ', precision_l1_opt, '\n',
      '                          for l2: ', precision_l2_opt)
print('       F1 score for unregularized: ', unreg_f1_score, '\n',
      '                          for l1: ', f1_score_l1_opt, '\n',
      '                          for l2: ', f1_score_l2_opt)
```

```
 Accuracy score for unregularized:  0.9714285714285714
                           for l1:  0.9704761904761905
                           for l2:  0.9701587301587301
   Recall score for unregularized:  0.9792452830188679
                           for l1:  0.9817610062893082
                           for l2:  0.9817610062893082
Precision score for unregularized:  0.9646840148698885
                           for l1:  0.9606153846153846
                           for l2:  0.9600246002460024
       F1 score for unregularized:  0.9719101123595505
                           for l1:  0.971073094867807
                           for l2:  0.9707711442786069
```

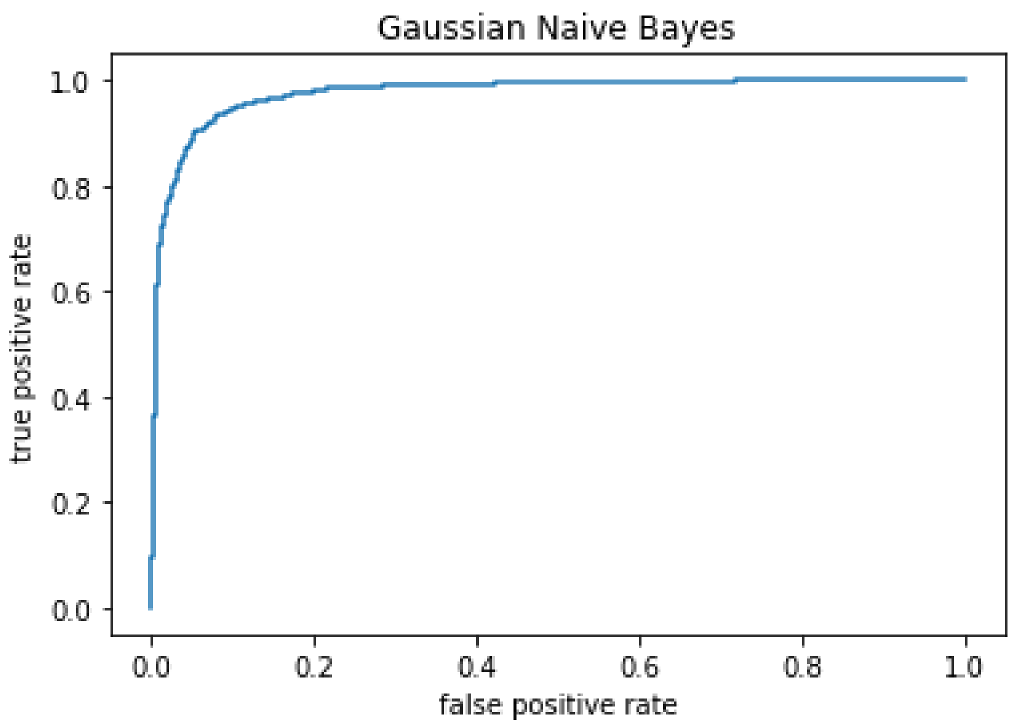
2.7 Quesiton 6

Do binary classification with (Gaussian) Naive Bayes. Fits a Gaussian probability model to the given features to estimate the classes. Fit and predict.

```
In [23]: from sklearn.naive_bayes import GaussianNB

         NBclsfr = GaussianNB()
         NBclsfr.fit(X_train_LSA, y_train)
         y_pred_nb = NBclsfr.predict(X_test_LSA)
         NB_scores_prob = NBclsfr.predict_proba(X_test_LSA)[:, 1]
```

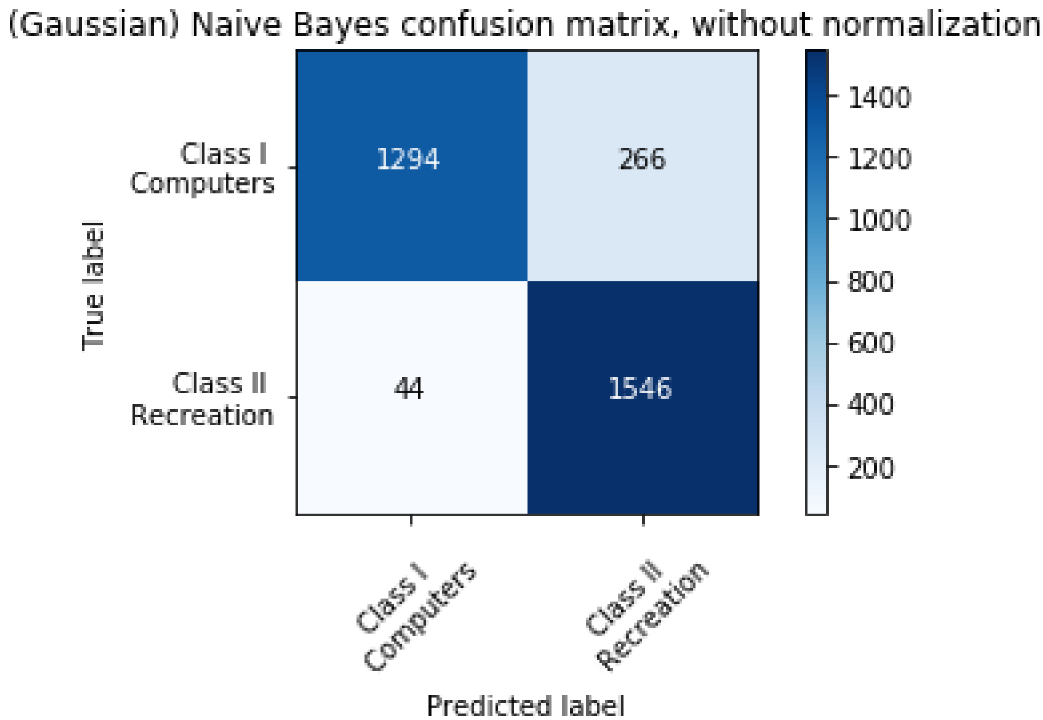
```
In [24]: # roc curve:
         fpr_gaussian_nb, tpr_gaussian_nb, thresholds_gaussian_nb = roc_curve(
             y_test, NB_scores_prob)

         # plot roc curves:
         plt.plot(fpr_gaussian_nb, tpr_gaussian_nb)
```

```
plt.title('Gaussian Naive Bayes')
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```




```
In [25]: confusion_nb = confusion_matrix(y_test, y_pred_nb)

         accuracy_nb = accuracy_score(y_test, y_pred_nb)
         recall_nb = recall_score(y_test, y_pred_nb)
         precision_nb = precision_score(y_test, y_pred_nb)
         f1_score_nb = f1_score(y_test, y_pred_nb)

         plot_confusion_matrix(confusion_nb, title='(Gaussian) Naive Bayes confusion matrix')

         print('Accuracy of (Gaussian) Naive Bayes:  ', accuracy_nb)
         print('Recall of (Gaussian) Naive Bayes:    ', recall_nb)
         print('Precision of (Gaussian) Naive Bayes: ', precision_nb)
         print('F1 score of (Gaussian) Naive Bayes:  ', f1_score_nb)
```

```
Accuracy of (Gaussian) Naive Bayes:   0.9015873015873016
Recall of (Gaussian) Naive Bayes:     0.9723270440251572
Precision of (Gaussian) Naive Bayes:  0.8532008830022075
F1 score of (Gaussian) Naive Bayes:   0.9088771310993533
```



2.8 Question 7

Grid search paramters to make it easier to optimize. - Construct a pipeline to f=perform feature extraction, dimensionality reduction and classification - Do grid search with 5-fold cross-validation to compare - removed headers and footers vs not - min_df = 3 or 5 - lemmatization or not - LSI or NMF - SVM with the previously found gamma or L1 log or L2 log or GaussianNB - else default

In [28]:
```
from sklearn.pipeline import Pipeline
from sklearn import linear_model
from sklearn.model_selection import GridSearchCV
import time

pipeline = Pipeline([
    ('vector', CountVectorizer(analyzer=rmv_nums, #toggle this and 'word'
                               min_df=3, #toggle min_df
                               stop_words='english')),
    ('tf-idf', TfidfTransformer()), # toggle this and  NMF
    ('reduce_dim', TruncatedSVD(n_components=50, random_state=0)),
    ('classify', GaussianNB()),     #toggle this and log reg's and SVM
])


param_grid = [
```

```
            {
                'vector__analyzer': [rmv_nums, 'word'], # lemmatizations vs not
                'vector__min_df': [3,5], # min df is 3 vs 5
                'reduce_dim': [TruncatedSVD(n_components=50, random_state=42),
                               NMF(n_components=50, init='random', random_state=42)],
                'classify': [GaussianNB(), # Gaussian Naive Bayes
                             LinearSVC(C=10, random_state=42), # SVC
                             LogisticRegression(penalty='l1', random_state=42, solver='libline
                             LogisticRegression(penalty='l2', random_state=42, solver='libline
            }
        ]
```

Create a small test parameter to grid to make sure GridSearch is performing as anticipated

```
In [32]: param_grid1 = [
             {
                  'vector__min_df': [3,5], # min df is 3 vs 5
             }
         ]
```

```
In [33]: grid = GridSearchCV(pipeline, cv=5, n_jobs=-1, param_grid=param_grid, verbose=5,
                             scoring='accuracy')
```

```
In [34]: start = time.time()
         grid.fit(train_dataset.data, y_train)
         end = time.time()
         print("run time is: ", end-start)
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed: 27.2min
[Parallel(n_jobs=-1)]: Done  64 tasks      | elapsed: 63.7min
[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed: 185.3min finished


run time is:  11222.44523692131
```

```
In [35]: import pandas as pd

         pd.DataFrame(grid.cv_results_)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
```

```
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
```

Out[35]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time \
0	195.302365	38.249045	771.261693	369.569553
1	123.171464	2.556296	30.887614	2.382495
2	3.369789	0.345350	0.459572	0.041375
3	2.871921	0.032609	0.431247	0.020926
4	184.331744	17.227365	31.777433	1.576711
5	156.060812	10.782643	30.578240	1.505317
6	43.637521	9.624553	0.601392	0.053139
7	36.378729	8.404199	0.582244	0.059980
8	121.637563	2.378637	30.792068	1.971063
9	120.769486	1.837846	31.181228	1.844527
10	3.062410	0.042208	0.446805	0.038909
11	2.859952	0.031353	0.451792	0.031240
12	154.900827	5.625651	30.588810	1.578923
13	150.328051	1.460439	30.299185	1.436035
14	42.715786	9.485130	0.598400	0.053750
15	35.644491	8.503039	0.555116	0.030327
16	120.717424	2.462456	29.612423	1.655945
17	119.308591	1.850170	29.571134	1.560754
18	3.037478	0.060779	0.439825	0.020016
19	2.882092	0.037638	0.423469	0.013420
20	153.752297	5.218158	30.340674	1.484078
21	149.098142	1.653014	30.146394	1.501141
22	44.323284	10.459883	0.599597	0.054771
23	35.472952	8.605531	0.544144	0.027499
24	886.714699	1528.236399	29.437291	1.016689
25	2411.709563	1877.037376	27.754789	1.365041
26	2.891866	0.026233	0.414890	0.013733
27	2.703970	0.057571	0.391154	0.015533
28	144.206022	5.483751	27.500468	1.339862
29	138.039310	1.596991	28.048004	1.618571
30	41.211408	9.218012	0.563693	0.049080
31	34.501948	7.990308	0.496272	0.074750

	param_classify \
0	GaussianNB(priors=None, var_smoothing=1e-09)
1	GaussianNB(priors=None, var_smoothing=1e-09)

```
2        GaussianNB(priors=None, var_smoothing=1e-09)
3        GaussianNB(priors=None, var_smoothing=1e-09)
4        GaussianNB(priors=None, var_smoothing=1e-09)
5        GaussianNB(priors=None, var_smoothing=1e-09)
6        GaussianNB(priors=None, var_smoothing=1e-09)
7        GaussianNB(priors=None, var_smoothing=1e-09)
8   LinearSVC(C=10, class_weight=None, dual=True, ...
9   LinearSVC(C=10, class_weight=None, dual=True, ...
10  LinearSVC(C=10, class_weight=None, dual=True, ...
11  LinearSVC(C=10, class_weight=None, dual=True, ...
12  LinearSVC(C=10, class_weight=None, dual=True, ...
13  LinearSVC(C=10, class_weight=None, dual=True, ...
14  LinearSVC(C=10, class_weight=None, dual=True, ...
15  LinearSVC(C=10, class_weight=None, dual=True, ...
16  LogisticRegression(C=10, class_weight=None, du...
17  LogisticRegression(C=10, class_weight=None, du...
18  LogisticRegression(C=10, class_weight=None, du...
19  LogisticRegression(C=10, class_weight=None, du...
20  LogisticRegression(C=10, class_weight=None, du...
21  LogisticRegression(C=10, class_weight=None, du...
22  LogisticRegression(C=10, class_weight=None, du...
23  LogisticRegression(C=10, class_weight=None, du...
24  LogisticRegression(C=100, class_weight=None, d...
25  LogisticRegression(C=100, class_weight=None, d...
26  LogisticRegression(C=100, class_weight=None, d...
27  LogisticRegression(C=100, class_weight=None, d...
28  LogisticRegression(C=100, class_weight=None, d...
29  LogisticRegression(C=100, class_weight=None, d...
30  LogisticRegression(C=100, class_weight=None, d...
31  LogisticRegression(C=100, class_weight=None, d...

                                     param_reduce_dim  \
0   TruncatedSVD(algorithm='randomized', n_compone...
1   TruncatedSVD(algorithm='randomized', n_compone...
2   TruncatedSVD(algorithm='randomized', n_compone...
3   TruncatedSVD(algorithm='randomized', n_compone...
4   NMF(alpha=0.0, beta_loss='frobenius', init='ra...
5   NMF(alpha=0.0, beta_loss='frobenius', init='ra...
6   NMF(alpha=0.0, beta_loss='frobenius', init='ra...
7   NMF(alpha=0.0, beta_loss='frobenius', init='ra...
8   TruncatedSVD(algorithm='randomized', n_compone...
9   TruncatedSVD(algorithm='randomized', n_compone...
10  TruncatedSVD(algorithm='randomized', n_compone...
11  TruncatedSVD(algorithm='randomized', n_compone...
12  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
13  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
14  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
15  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
```

```
16  TruncatedSVD(algorithm='randomized', n_compone...
17  TruncatedSVD(algorithm='randomized', n_compone...
18  TruncatedSVD(algorithm='randomized', n_compone...
19  TruncatedSVD(algorithm='randomized', n_compone...
20  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
21  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
22  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
23  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
24  TruncatedSVD(algorithm='randomized', n_compone...
25  TruncatedSVD(algorithm='randomized', n_compone...
26  TruncatedSVD(algorithm='randomized', n_compone...
27  TruncatedSVD(algorithm='randomized', n_compone...
28  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
29  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
30  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
31  NMF(alpha=0.0, beta_loss='frobenius', init='ra...

                       param_vector__analyzer param_vector__min_df  \
0   <function rmv_nums at 0x00000174B45B6A60>                    3
1   <function rmv_nums at 0x00000174B45B6A60>                    5
2                                        word                    3
3                                        word                    5
4   <function rmv_nums at 0x00000174B45B6A60>                    3
5   <function rmv_nums at 0x00000174B45B6A60>                    5
6                                        word                    3
7                                        word                    5
8   <function rmv_nums at 0x00000174B45B6A60>                    3
9   <function rmv_nums at 0x00000174B45B6A60>                    5
10                                       word                    3
11                                       word                    5
12  <function rmv_nums at 0x00000174B45B6A60>                    3
13  <function rmv_nums at 0x00000174B45B6A60>                    5
14                                       word                    3
15                                       word                    5
16  <function rmv_nums at 0x00000174B45B6A60>                    3
17  <function rmv_nums at 0x00000174B45B6A60>                    5
18                                       word                    3
19                                       word                    5
20  <function rmv_nums at 0x00000174B45B6A60>                    3
21  <function rmv_nums at 0x00000174B45B6A60>                    5
22                                       word                    3
23                                       word                    5
24  <function rmv_nums at 0x00000174B45B6A60>                    3
25  <function rmv_nums at 0x00000174B45B6A60>                    5
26                                       word                    3
27                                       word                    5
28  <function rmv_nums at 0x00000174B45B6A60>                    3
29  <function rmv_nums at 0x00000174B45B6A60>                    5
```

```
30                                      word                  3
31                                      word                  5

                                                params  split0_test_score  \
0   {'classify': GaussianNB(priors=None, var_smoot...           0.928194
1   {'classify': GaussianNB(priors=None, var_smoot...           0.920803
2   {'classify': GaussianNB(priors=None, var_smoot...           0.884900
3   {'classify': GaussianNB(priors=None, var_smoot...           0.920803
4   {'classify': GaussianNB(priors=None, var_smoot...           0.936642
5   {'classify': GaussianNB(priors=None, var_smoot...           0.952482
6   {'classify': GaussianNB(priors=None, var_smoot...           0.949314
7   {'classify': GaussianNB(priors=None, var_smoot...           0.951426
8   {'classify': LinearSVC(C=10, class_weight=None...           0.971489
9   {'classify': LinearSVC(C=10, class_weight=None...           0.972545
10  {'classify': LinearSVC(C=10, class_weight=None...           0.967265
11  {'classify': LinearSVC(C=10, class_weight=None...           0.968321
12  {'classify': LinearSVC(C=10, class_weight=None...           0.964097
13  {'classify': LinearSVC(C=10, class_weight=None...           0.968321
14  {'classify': LinearSVC(C=10, class_weight=None...           0.969377
15  {'classify': LinearSVC(C=10, class_weight=None...           0.969377
16  {'classify': LogisticRegression(C=10, class_we...           0.970433
17  {'classify': LogisticRegression(C=10, class_we...           0.972545
18  {'classify': LogisticRegression(C=10, class_we...           0.972545
19  {'classify': LogisticRegression(C=10, class_we...           0.971489
20  {'classify': LogisticRegression(C=10, class_we...           0.963041
21  {'classify': LogisticRegression(C=10, class_we...           0.968321
22  {'classify': LogisticRegression(C=10, class_we...           0.965153
23  {'classify': LogisticRegression(C=10, class_we...           0.961985
24  {'classify': LogisticRegression(C=100, class_w...           0.972545
25  {'classify': LogisticRegression(C=100, class_w...           0.971489
26  {'classify': LogisticRegression(C=100, class_w...           0.970433
27  {'classify': LogisticRegression(C=100, class_w...           0.970433
28  {'classify': LogisticRegression(C=100, class_w...           0.964097
29  {'classify': LogisticRegression(C=100, class_w...           0.969377
30  {'classify': LogisticRegression(C=100, class_w...           0.968321
31  {'classify': LogisticRegression(C=100, class_w...           0.967265

         ...          mean_test_score  std_test_score  rank_test_score  \
0        ...                 0.898563        0.042416               31
1        ...                 0.880389        0.048132               32
2        ...                 0.912933        0.015050               30
3        ...                 0.913145        0.014795               29
4        ...                 0.938081        0.006604               28
5        ...                 0.942096        0.005966               26
6        ...                 0.943576        0.005214               25
7        ...                 0.939772        0.007733               27
8        ...                 0.975063        0.004566                1
9        ...                 0.973795        0.005241                8
```

10	...	0.974218	0.004084	6
11	...	0.972527	0.004374	12
12	...	0.963652	0.003630	24
13	...	0.965342	0.005332	22
14	...	0.966822	0.005228	16
15	...	0.965554	0.003568	19
16	...	0.974641	0.004483	4
17	...	0.975063	0.004565	1
18	...	0.974641	0.003192	4
19	...	0.973584	0.003592	10
20	...	0.969992	0.004344	13
21	...	0.969569	0.005240	14
22	...	0.967878	0.003227	15
23	...	0.965554	0.003919	19
24	...	0.975063	0.004759	1
25	...	0.973795	0.004933	8
26	...	0.974218	0.003369	6
27	...	0.972739	0.002932	11
28	...	0.964497	0.003019	23
29	...	0.965554	0.005387	19
30	...	0.965765	0.004639	18
31	...	0.966610	0.003102	17

	split0_train_score	split1_train_score	split2_train_score	\
0	0.929458	0.906737	0.842272	
1	0.917834	0.885337	0.833554	
2	0.904888	0.928402	0.913342	
3	0.923910	0.919947	0.896697	
4	0.936856	0.937649	0.942668	
5	0.944782	0.940026	0.937120	
6	0.944782	0.946631	0.942404	
7	0.947952	0.940819	0.933421	
8	0.979128	0.976222	0.978071	
9	0.978336	0.975958	0.978071	
10	0.978336	0.978336	0.976750	
11	0.978600	0.977015	0.976750	
12	0.961162	0.966711	0.964333	
13	0.967768	0.962219	0.964333	
14	0.968824	0.964597	0.964333	
15	0.968824	0.964069	0.962748	
16	0.978071	0.976486	0.978600	
17	0.978336	0.976486	0.978071	
18	0.978071	0.978600	0.976486	
19	0.978864	0.978600	0.977279	
20	0.965125	0.971466	0.969617	
21	0.974108	0.968296	0.972787	
22	0.970410	0.968824	0.967768	
23	0.972259	0.968032	0.963804	

```
24           0.977543           0.976486           0.977543
25           0.978336           0.975958           0.978071
26           0.978336           0.978864           0.976486
27           0.978071           0.977807           0.976486
28           0.960634           0.966711           0.965125
29           0.967503           0.963276           0.964333
30           0.969353           0.965125           0.964861
31           0.968032           0.963540           0.963012

    split3_train_score  split4_train_score  mean_train_score  std_train_score
0             0.907818            0.930024          0.903262         0.032109
1             0.873217            0.924743          0.886937         0.032938
2             0.912044            0.894904          0.910716         0.011004
3             0.886424            0.918669          0.909130         0.014809
4             0.938193            0.942171          0.939507         0.002421
5             0.937929            0.933721          0.938716         0.003651
6             0.946117            0.942435          0.944474         0.001783
7             0.942684            0.935833          0.940142         0.005131
8             0.980718            0.978083          0.978445         0.001473
9             0.979398            0.978875          0.978127         0.001177
10            0.979926            0.978875          0.978445         0.001027
11            0.978341            0.978347          0.977811         0.000768
12            0.963814            0.961711          0.963546         0.001988
13            0.964871            0.963031          0.964444         0.001907
14            0.961965            0.963295          0.964603         0.002305
15            0.964871            0.965408          0.965184         0.002029
16            0.980190            0.979403          0.978550         0.001258
17            0.980454            0.978875          0.978444         0.001281
18            0.980718            0.977819          0.978339         0.001379
19            0.979926            0.976234          0.978181         0.001288
20            0.968568            0.966200          0.968195         0.002292
21            0.971474            0.968577          0.971048         0.002291
22            0.966455            0.967520          0.968195         0.001339
23            0.969625            0.970689          0.968882         0.002889
24            0.979398            0.979139          0.978022         0.001092
25            0.979398            0.978611          0.978075         0.001148
26            0.979134            0.978875          0.978339         0.000962
27            0.978605            0.978347          0.977863         0.000739
28            0.964078            0.961447          0.963599         0.002266
29            0.965663            0.962503          0.964656         0.001776
30            0.963022            0.965672          0.965607         0.002074
31            0.964606            0.965936          0.965025         0.001806

[32 rows x 24 columns]
```

Remove header and footer from Data

In [38]:
```
train_dataset_noheaders = fetch_20newsgroups(subset = 'train',
                                             categories=comp_categories+rec_categories,
```

```
                                             remove=('headers', 'footers'), #toggle head/foot
                                             shuffle = True,
                                             random_state = None)
        test_dataset_noheaders = fetch_20newsgroups(subset = 'test',
                                            categories=comp_categories+rec_categories,
                                            remove=('headers', 'footers'), #toggle head/foot
                                            shuffle = True,
                                            random_state = None)
```

In [39]:
```
y_train_noheaders = train_dataset_noheaders.target > 3
y_test_noheaders = test_dataset_noheaders.target > 3
```

In [40]:
```
grid1 = GridSearchCV(pipeline, cv=5, n_jobs=-1, param_grid=param_grid, verbose=5, sco
start = time.time()
grid1.fit(train_dataset_noheaders.data, y_train_noheaders)
end = time.time()
print("run time is: ", end-start)
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  8.0min
[Parallel(n_jobs=-1)]: Done  64 tasks      | elapsed: 116.9min
[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed: 246.5min finished


run time is:  14792.309785842896
```

In [42]:
```
import pandas as pd

pd.DataFrame(grid.cv_results_)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
```

Out[42]:
```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0      195.302365     38.249045       771.261693      369.569553
1      123.171464      2.556296        30.887614        2.382495
2        3.369789      0.345350         0.459572        0.041375
3        2.871921      0.032609         0.431247        0.020926
4      184.331744     17.227365        31.777433        1.576711
5      156.060812     10.782643        30.578240        1.505317
6       43.637521      9.624553         0.601392        0.053139
7       36.378729      8.404199         0.582244        0.059980
8      121.637563      2.378637        30.792068        1.971063
9      120.769486      1.837846        31.181228        1.844527
10       3.062410      0.042208         0.446805        0.038909
11       2.859952      0.031353         0.451792        0.031240
12     154.900827      5.625651        30.588810        1.578923
13     150.328051      1.460439        30.299185        1.436035
14      42.715786      9.485130         0.598400        0.053750
15      35.644491      8.503039         0.555116        0.030327
16     120.717424      2.462456        29.612423        1.655945
17     119.308591      1.850170        29.571134        1.560754
18       3.037478      0.060779         0.439825        0.020016
19       2.882092      0.037638         0.423469        0.013420
20     153.752297      5.218158        30.340674        1.484078
21     149.098142      1.653014        30.146394        1.501141
22      44.323284     10.459883         0.599597        0.054771
23      35.472952      8.605531         0.544144        0.027499
24     886.714699   1528.236399        29.437291        1.016689
25    2411.709563   1877.037376        27.754789        1.365041
26       2.891866      0.026233         0.414890        0.013733
27       2.703970      0.057571         0.391154        0.015533
28     144.206022      5.483751        27.500468        1.339862
29     138.039310      1.596991        28.048004        1.618571
30      41.211408      9.218012         0.563693        0.049080
31      34.501948      7.990308         0.496272        0.074750

                                       param_classify  \
0        GaussianNB(priors=None, var_smoothing=1e-09)
1        GaussianNB(priors=None, var_smoothing=1e-09)
2        GaussianNB(priors=None, var_smoothing=1e-09)
3        GaussianNB(priors=None, var_smoothing=1e-09)
4        GaussianNB(priors=None, var_smoothing=1e-09)
5        GaussianNB(priors=None, var_smoothing=1e-09)
6        GaussianNB(priors=None, var_smoothing=1e-09)
7        GaussianNB(priors=None, var_smoothing=1e-09)
8   LinearSVC(C=10, class_weight=None, dual=True, ...
9   LinearSVC(C=10, class_weight=None, dual=True, ...
10  LinearSVC(C=10, class_weight=None, dual=True, ...
11  LinearSVC(C=10, class_weight=None, dual=True, ...
12  LinearSVC(C=10, class_weight=None, dual=True, ...
```

```
13  LinearSVC(C=10, class_weight=None, dual=True, ...
14  LinearSVC(C=10, class_weight=None, dual=True, ...
15  LinearSVC(C=10, class_weight=None, dual=True, ...
16  LogisticRegression(C=10, class_weight=None, du...
17  LogisticRegression(C=10, class_weight=None, du...
18  LogisticRegression(C=10, class_weight=None, du...
19  LogisticRegression(C=10, class_weight=None, du...
20  LogisticRegression(C=10, class_weight=None, du...
21  LogisticRegression(C=10, class_weight=None, du...
22  LogisticRegression(C=10, class_weight=None, du...
23  LogisticRegression(C=10, class_weight=None, du...
24  LogisticRegression(C=100, class_weight=None, d...
25  LogisticRegression(C=100, class_weight=None, d...
26  LogisticRegression(C=100, class_weight=None, d...
27  LogisticRegression(C=100, class_weight=None, d...
28  LogisticRegression(C=100, class_weight=None, d...
29  LogisticRegression(C=100, class_weight=None, d...
30  LogisticRegression(C=100, class_weight=None, d...
31  LogisticRegression(C=100, class_weight=None, d...

                                   param_reduce_dim  \
0   TruncatedSVD(algorithm='randomized', n_compone...
1   TruncatedSVD(algorithm='randomized', n_compone...
2   TruncatedSVD(algorithm='randomized', n_compone...
3   TruncatedSVD(algorithm='randomized', n_compone...
4   NMF(alpha=0.0, beta_loss='frobenius', init='ra...
5   NMF(alpha=0.0, beta_loss='frobenius', init='ra...
6   NMF(alpha=0.0, beta_loss='frobenius', init='ra...
7   NMF(alpha=0.0, beta_loss='frobenius', init='ra...
8   TruncatedSVD(algorithm='randomized', n_compone...
9   TruncatedSVD(algorithm='randomized', n_compone...
10  TruncatedSVD(algorithm='randomized', n_compone...
11  TruncatedSVD(algorithm='randomized', n_compone...
12  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
13  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
14  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
15  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
16  TruncatedSVD(algorithm='randomized', n_compone...
17  TruncatedSVD(algorithm='randomized', n_compone...
18  TruncatedSVD(algorithm='randomized', n_compone...
19  TruncatedSVD(algorithm='randomized', n_compone...
20  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
21  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
22  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
23  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
24  TruncatedSVD(algorithm='randomized', n_compone...
25  TruncatedSVD(algorithm='randomized', n_compone...
26  TruncatedSVD(algorithm='randomized', n_compone...
```

```
27  TruncatedSVD(algorithm='randomized', n_compone...
28  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
29  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
30  NMF(alpha=0.0, beta_loss='frobenius', init='ra...
31  NMF(alpha=0.0, beta_loss='frobenius', init='ra...

                      param_vector__analyzer param_vector__min_df  \
0   <function rmv_nums at 0x00000174B45B6A60>                    3
1   <function rmv_nums at 0x00000174B45B6A60>                    5
2                                        word                    3
3                                        word                    5
4   <function rmv_nums at 0x00000174B45B6A60>                    3
5   <function rmv_nums at 0x00000174B45B6A60>                    5
6                                        word                    3
7                                        word                    5
8   <function rmv_nums at 0x00000174B45B6A60>                    3
9   <function rmv_nums at 0x00000174B45B6A60>                    5
10                                       word                    3
11                                       word                    5
12  <function rmv_nums at 0x00000174B45B6A60>                    3
13  <function rmv_nums at 0x00000174B45B6A60>                    5
14                                       word                    3
15                                       word                    5
16  <function rmv_nums at 0x00000174B45B6A60>                    3
17  <function rmv_nums at 0x00000174B45B6A60>                    5
18                                       word                    3
19                                       word                    5
20  <function rmv_nums at 0x00000174B45B6A60>                    3
21  <function rmv_nums at 0x00000174B45B6A60>                    5
22                                       word                    3
23                                       word                    5
24  <function rmv_nums at 0x00000174B45B6A60>                    3
25  <function rmv_nums at 0x00000174B45B6A60>                    5
26                                       word                    3
27                                       word                    5
28  <function rmv_nums at 0x00000174B45B6A60>                    3
29  <function rmv_nums at 0x00000174B45B6A60>                    5
30                                       word                    3
31                                       word                    5

                                               params  split0_test_score  \
0   {'classify': GaussianNB(priors=None, var_smoot...           0.928194
1   {'classify': GaussianNB(priors=None, var_smoot...           0.920803
2   {'classify': GaussianNB(priors=None, var_smoot...           0.884900
3   {'classify': GaussianNB(priors=None, var_smoot...           0.920803
4   {'classify': GaussianNB(priors=None, var_smoot...           0.936642
5   {'classify': GaussianNB(priors=None, var_smoot...           0.952482
6   {'classify': GaussianNB(priors=None, var_smoot...           0.949314
```

```
7   {'classify': GaussianNB(priors=None, var_smoot...          0.951426
8   {'classify': LinearSVC(C=10, class_weight=None...          0.971489
9   {'classify': LinearSVC(C=10, class_weight=None...          0.972545
10  {'classify': LinearSVC(C=10, class_weight=None...          0.967265
11  {'classify': LinearSVC(C=10, class_weight=None...          0.968321
12  {'classify': LinearSVC(C=10, class_weight=None...          0.964097
13  {'classify': LinearSVC(C=10, class_weight=None...          0.968321
14  {'classify': LinearSVC(C=10, class_weight=None...          0.969377
15  {'classify': LinearSVC(C=10, class_weight=None...          0.969377
16  {'classify': LogisticRegression(C=10, class_we...          0.970433
17  {'classify': LogisticRegression(C=10, class_we...          0.972545
18  {'classify': LogisticRegression(C=10, class_we...          0.972545
19  {'classify': LogisticRegression(C=10, class_we...          0.971489
20  {'classify': LogisticRegression(C=10, class_we...          0.963041
21  {'classify': LogisticRegression(C=10, class_we...          0.968321
22  {'classify': LogisticRegression(C=10, class_we...          0.965153
23  {'classify': LogisticRegression(C=10, class_we...          0.961985
24  {'classify': LogisticRegression(C=100, class_w...          0.972545
25  {'classify': LogisticRegression(C=100, class_w...          0.971489
26  {'classify': LogisticRegression(C=100, class_w...          0.970433
27  {'classify': LogisticRegression(C=100, class_w...          0.970433
28  {'classify': LogisticRegression(C=100, class_w...          0.964097
29  {'classify': LogisticRegression(C=100, class_w...          0.969377
30  {'classify': LogisticRegression(C=100, class_w...          0.968321
31  {'classify': LogisticRegression(C=100, class_w...          0.967265

         ...          mean_test_score  std_test_score  rank_test_score  \
0        ...                 0.898563        0.042416               31
1        ...                 0.880389        0.048132               32
2        ...                 0.912933        0.015050               30
3        ...                 0.913145        0.014795               29
4        ...                 0.938081        0.006604               28
5        ...                 0.942096        0.005966               26
6        ...                 0.943576        0.005214               25
7        ...                 0.939772        0.007733               27
8        ...                 0.975063        0.004566                1
9        ...                 0.973795        0.005241                8
10       ...                 0.974218        0.004084                6
11       ...                 0.972527        0.004374               12
12       ...                 0.963652        0.003630               24
13       ...                 0.965342        0.005332               22
14       ...                 0.966822        0.005228               16
15       ...                 0.965554        0.003568               19
16       ...                 0.974641        0.004483                4
17       ...                 0.975063        0.004565                1
18       ...                 0.974641        0.003192                4
19       ...                 0.973584        0.003592               10
20       ...                 0.969992        0.004344               13
```

```
21       ...           0.969569        0.005240              14
22       ...           0.967878        0.003227              15
23       ...           0.965554        0.003919              19
24       ...           0.975063        0.004759               1
25       ...           0.973795        0.004933               8
26       ...           0.974218        0.003369               6
27       ...           0.972739        0.002932              11
28       ...           0.964497        0.003019              23
29       ...           0.965554        0.005387              19
30       ...           0.965765        0.004639              18
31       ...           0.966610        0.003102              17

    split0_train_score  split1_train_score  split2_train_score  \
0             0.929458            0.906737            0.842272
1             0.917834            0.885337            0.833554
2             0.904888            0.928402            0.913342
3             0.923910            0.919947            0.896697
4             0.936856            0.937649            0.942668
5             0.944782            0.940026            0.937120
6             0.944782            0.946631            0.942404
7             0.947952            0.940819            0.933421
8             0.979128            0.976222            0.978071
9             0.978336            0.975958            0.978071
10            0.978336            0.978336            0.976750
11            0.978600            0.977015            0.976750
12            0.961162            0.966711            0.964333
13            0.967768            0.962219            0.964333
14            0.968824            0.964597            0.964333
15            0.968824            0.964069            0.962748
16            0.978071            0.976486            0.978600
17            0.978336            0.976486            0.978071
18            0.978071            0.978600            0.976486
19            0.978864            0.978600            0.977279
20            0.965125            0.971466            0.969617
21            0.974108            0.968296            0.972787
22            0.970410            0.968824            0.967768
23            0.972259            0.968032            0.963804
24            0.977543            0.976486            0.977543
25            0.978336            0.975958            0.978071
26            0.978336            0.978864            0.976486
27            0.978071            0.977807            0.976486
28            0.960634            0.966711            0.965125
29            0.967503            0.963276            0.964333
30            0.969353            0.965125            0.964861
31            0.968032            0.963540            0.963012

    split3_train_score  split4_train_score  mean_train_score  std_train_score
0             0.907818            0.930024          0.903262         0.032109
```

1	0.873217	0.924743	0.886937	0.032938
2	0.912044	0.894904	0.910716	0.011004
3	0.886424	0.918669	0.909130	0.014809
4	0.938193	0.942171	0.939507	0.002421
5	0.937929	0.933721	0.938716	0.003651
6	0.946117	0.942435	0.944474	0.001783
7	0.942684	0.935833	0.940142	0.005131
8	0.980718	0.978083	0.978445	0.001473
9	0.979398	0.978875	0.978127	0.001177
10	0.979926	0.978875	0.978445	0.001027
11	0.978341	0.978347	0.977811	0.000768
12	0.963814	0.961711	0.963546	0.001988
13	0.964871	0.963031	0.964444	0.001907
14	0.961965	0.963295	0.964603	0.002305
15	0.964871	0.965408	0.965184	0.002029
16	0.980190	0.979403	0.978550	0.001258
17	0.980454	0.978875	0.978444	0.001281
18	0.980718	0.977819	0.978339	0.001379
19	0.979926	0.976234	0.978181	0.001288
20	0.968568	0.966200	0.968195	0.002292
21	0.971474	0.968577	0.971048	0.002291
22	0.966455	0.967520	0.968195	0.001339
23	0.969625	0.970689	0.968882	0.002889
24	0.979398	0.979139	0.978022	0.001092
25	0.979398	0.978611	0.978075	0.001148
26	0.979134	0.978875	0.978339	0.000962
27	0.978605	0.978347	0.977863	0.000739
28	0.964078	0.961447	0.963599	0.002266
29	0.965663	0.962503	0.964656	0.001776
30	0.963022	0.965672	0.965607	0.002074
31	0.964606	0.965936	0.965025	0.001806

```
[32 rows x 24 columns]
```

3 Should this be removed?

Create GridSearch object, Use param_grid_small for testing!!!!!!!! only use param_grid if you're ready to brick your computer for hours.

4 Should this be removed?

fun_classifier = GridSearchCV(pipeline, cv=2, n_jobs=-1, param_grid=param_grid,verbose=5, scoring='accuracy')

5 print(fun_classifier.cv)

6 dir(fun_classifier)

7 Should this be removed?

Load fresh data

8 Should this be removed?

categories = ['comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey'] train_dataset = fetch_20newsgroups(subset = 'train', categories = categories, shuffle = True, random_state = 42) test_dataset = fetch_20newsgroups(subset = 'test', categories = categories, shuffle = True, random_state = 42)

9 Should this be removed?

Run GridSearch fitting (warning: very long run time) - Just 1 set of paramters with cv=2, n_jobs=1 -> 8+ minutes on my computer - slightly faster with n_jobs=-1 (use as much parallization as possible). will explode your cpu % so use n_jobs=1 if you dont want that

10 Should this be removed?

import time start=time.time() fun_classifier.fit(train_dataset.data, train_dataset.target) end=time.time() print("runtime:",end-start)

11 Should this be removed?

12 attempt to save the above results to disk....

from sklearn.externals import joblib # dump to pickle joblib.dump(fun_classifier, 'grid_result1.pkl')

13 Should this be removed?

14 and reload from pickle

fun_classifier = joblib.load('grid_result1.pkl')

15 Should this be removed?

print('-'*40)
print('Best Index:', fun_classifier.best_index_)

print('Best Params:', fun_classifier.best_params_)
print('Best score:', fun_classifier.best_score_)

16 Should this be removed?

17 fun_classifier.predict(test_dataset.data)

Redo with headers and footers removed

18 Should this be removed?

import time start=time.time() fun_classifier_nohead = GridSearchCV(pipeline, cv=2, n_jobs=-1, param_grid=param_grid,verbose=5, scoring='accuracy') fun_classifier_nohead.fit(train_dataset.data, train_dataset.target) end=time.time() print("runtime:",end-start)

19 Should this be removed?

print('-'*40)
print('Best Index:', fun_classifier_nohead.best_index_)
print('Best Params:', fun_classifier_nohead.best_params_)
print('Best score:', fun_classifier_nohead.best_score_)

20 Should this be removed?

21 attempt to save the above results to disk....

from sklearn.externals import joblib # dump to pickle joblib.dump(fun_classifier_nohead, 'grid_result_2.pkl')

22 Should this be removed?

import pandas as pd
pd.DataFrame(fun_classifier.cv_results_)

23 Should this be removed?

import pandas as pd
pd.DataFrame(fun_classifier_nohead.cv_results_)

24 Question 8

```
In [43]: categories_q8 = ['comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
                          'misc.forsale', 'soc.religion.christian']
```

```
In [46]: train_dataset_q8 = fetch_20newsgroups(subset = 'train', categories = categories_q8,
                                               shuffle = True, random_state = None)
         test_dataset_q8 = fetch_20newsgroups(subset = 'test', categories = categories_q8,
                                              shuffle = True, random_state = None)
```

24.0.1 Feature Extraction

```
In [74]: X_train_q8_counts = vectorizer.fit_transform(train_dataset_q8.data)
         X_test_q8_counts = vectorizer.transform(test_dataset_q8.data)
         print('Shape of training data matrix:  ',X_train_q8_counts.shape)
         print('Shape of testing data matrix:   ',X_test_q8_counts.shape)
```

```
Shape of training data matrix:   (2352, 8699)
Shape of testing data matrix:    (1565, 8699)
```

```
In [48]: tfidf_transformer = TfidfTransformer()
         X_train_q8_tfidf = tfidf_transformer.fit_transform(X_train_q8_counts)
         X_test_q8_tfidf = tfidf_transformer.transform(X_test_q8_counts)
```

24.0.2 Dimentionality Reduction, LSA

```
In [75]: svd = TruncatedSVD(n_components=50, random_state=42)
         X_train_q8_lsa = svd.fit_transform(X_train_q8_tfidf)
         X_test_q8_lsa = svd.transform(X_test_q8_tfidf)
         print('Shape of LSA training data matrix:  ',X_train_q8_lsa.shape)
         print('Shape of LSA testing data matrix:   ',X_test_q8_lsa.shape)
```

```
Shape of LSA training data matrix:   (2352, 50)
Shape of LSA testing data matrix:    (1565, 50)
```

24.0.3 Naive Bayes

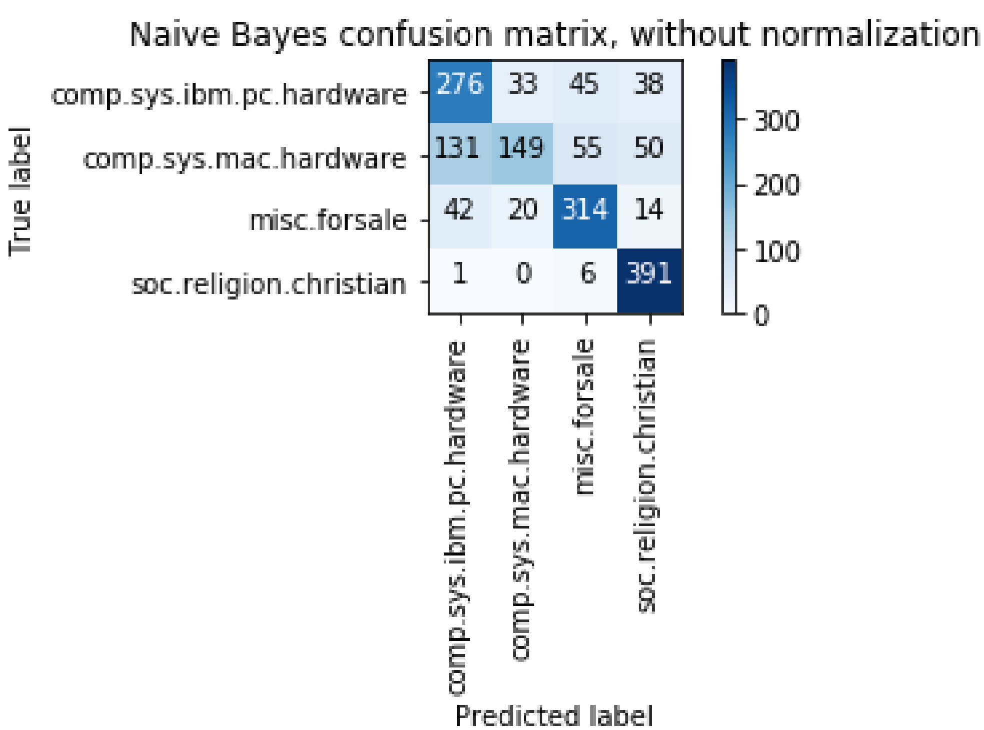
```
In [50]: clf_multiclass_nb = GaussianNB()
         clf_multiclass_nb.fit(X_train_q8_lsa,train_dataset_q8.target)
         y_pred_q8_nb = clf_multiclass_nb.predict(X_test_q8_lsa)
         scores_prob_q8_nb = np.argmax(clf_multiclass_nb.predict_proba(X_test_q8_lsa), 1)
```

```
In [51]: confusion_q8_nb = confusion_matrix(test_dataset_q8.target, y_pred_q8_nb)
```

```
In [85]: #4-dim confusion matrix plot
         import itertools
```

```python
def plot_confusion_matrix_4(cm, title, cmap = plt.cm.Blues):
    ctgrs = ['comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
                  'misc.forsale', 'soc.religion.christian']
    plt.figure()
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title + ', without normalization')
    plt.colorbar()
    tick_marks = np.arange(len(ctgrs))
    plt.xticks(tick_marks, ctgrs, rotation=90)
    plt.yticks(tick_marks, ctgrs)

    fmt = 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

```python
plot_confusion_matrix_4(confusion_q8_nb, title='Naive Bayes confusion matrix')
```



```
In [55]: accuracy_q8_nb = accuracy_score(test_dataset_q8.target, y_pred_q8_nb)
         recall_q8_nb = recall_score(test_dataset_q8.target, y_pred_q8_nb, average='weighted')
         precision_q8_nb = precision_score(test_dataset_q8.target, y_pred_q8_nb, average='weig
         f1_score_q8_nb = f1_score(test_dataset_q8.target, y_pred_q8_nb, average='weighted')

         print('Accuracy logistic:  ', accuracy_q8_nb) #
         print('Recall logistic:    ', recall_q8_nb) #
         print('Precision logistic: ', precision_q8_nb) #
         print('F1 score logistic:  ', f1_score_q8_nb) #
```

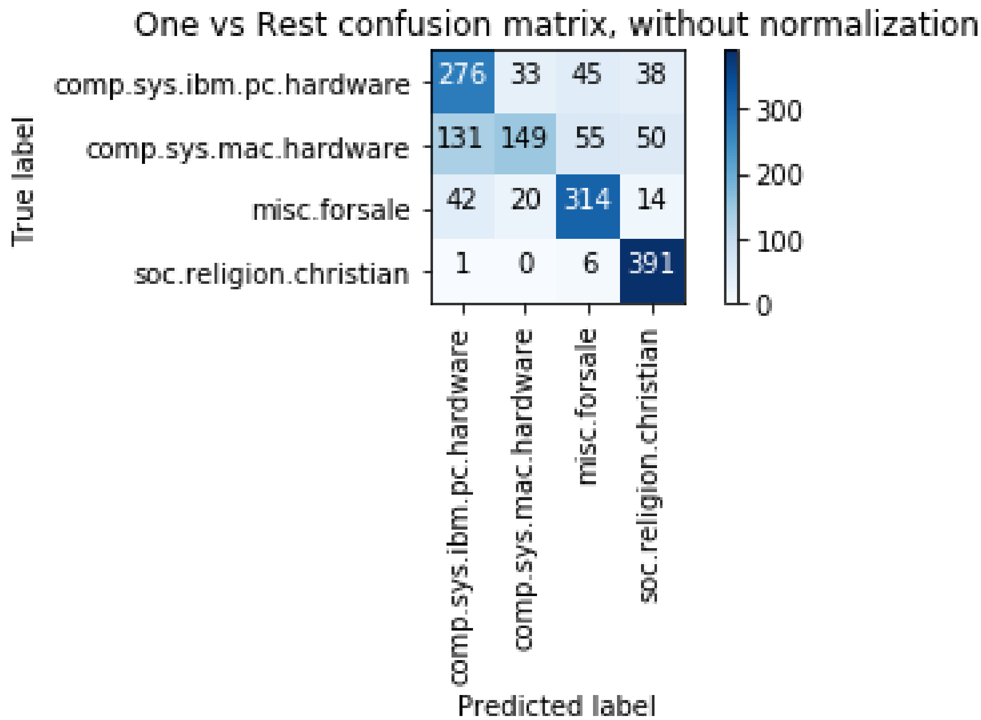
```
Accuracy logistic:   0.7220447284345048
Recall logistic:     0.7220447284345048
Precision logistic:  0.7230916397231564
F1 score logistic:   0.7055085166587322
```

24.0.4 Multi class SVM

One vs Rest

```
In [87]: from sklearn.svm import SVC
         from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier

         clf_multiclass_1vsR = OneVsRestClassifier(LinearSVC(random_state=42, C=1))
         clf_multiclass_1vsR.fit(X_train_q8_lsa, train_dataset_q8.target)
         y_pred_q8_multiclass_1vsR = clf_multiclass_1vsR.predict(X_test_q8_lsa)
```

```
In [89]: confusion_q8_multiclass_1vsR = confusion_matrix(test_dataset_q8.target,
                                                         y_pred_q8_multiclass_1vsR)
         plot_confusion_matrix_4(confusion_q8_nb, title='One vs Rest confusion matrix')
```



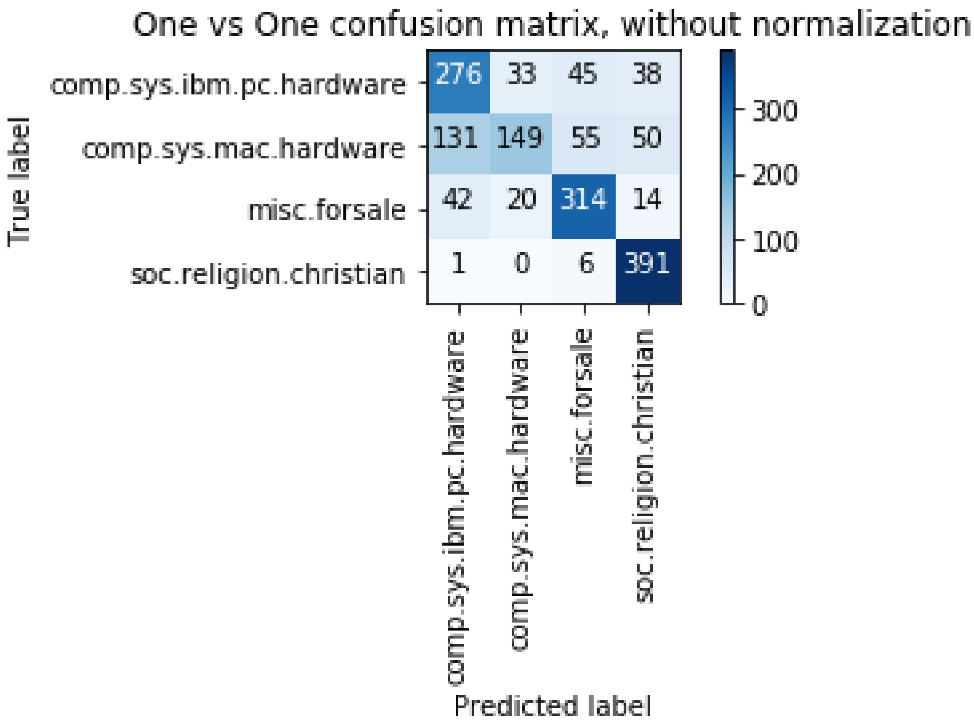

```
In [86]: accuracy_q8_1vsR = accuracy_score(test_dataset_q8.target, y_pred_q8_multiclass_1vsR)
         recall_q8_1vsR = recall_score(test_dataset_q8.target, y_pred_q8_multiclass_1vsR, avera
         precision_q8_1vsR = precision_score(test_dataset_q8.target, y_pred_q8_multiclass_1vsR
         f1_score_q8_1vsR = f1_score(test_dataset_q8.target, y_pred_q8_multiclass_1vsR, average

         print('Accuracy logistic:  ', accuracy_q8_1vsR)
         print('Recall logistic:    ', recall_q8_1vsR)
         print('Precision logistic: ', precision_q8_1vsR)
         print('F1 score logistic:  ', f1_score_q8_1vsR)
```

```
Accuracy logistic:   0.8817891373801917
Recall logistic:     0.8817891373801917
Precision logistic:  0.8810351309098071
F1 score logistic:   0.8812616340887313
```

One vs One

```
In [88]: clf_multiclass_1vs1 = OneVsOneClassifier(LinearSVC(random_state=42))
         clf_multiclass_1vs1.fit(X_train_q8_lsa, train_dataset_q8.target)
         y_pred_q8_multiclass_1vs1 = clf_multiclass_1vs1.predict(X_test_q8_lsa)
```

```
In [91]: confusion_q8_multiclass_1vs1 = confusion_matrix(test_dataset_q8.target,
                                                         y_pred_q8_multiclass_1vs1)
         plot_confusion_matrix_4(confusion_q8_nb, title='One vs One confusion matrix')
```




```
In [93]: accuracy_q8_1vs1 = accuracy_score(test_dataset_q8.target, y_pred_q8_multiclass_1vs1)
         recall_q8_1vs1 = recall_score(test_dataset_q8.target, y_pred_q8_multiclass_1vs1, avera
         precision_q8_1vs1 = precision_score(test_dataset_q8.target, y_pred_q8_multiclass_1vs1
         f1_score_q8_1vs1 = f1_score(test_dataset_q8.target, y_pred_q8_multiclass_1vs1, average

         print('Accuracy logistic:  ', accuracy_q8_1vs1)
         print('Recall logistic:    ', recall_q8_1vs1)
         print('Precision logistic: ', precision_q8_1vs1)
         print('F1 score logistic:  ', f1_score_q8_1vs1)
```

```
Accuracy logistic:   0.8849840255591054
Recall logistic:     0.8849840255591054
Precision logistic:  0.8855504919847299
F1 score logistic:   0.8851747756095596
```

```
In [ ]:
```