```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from nltk.corpus import stopwords
from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB

import time

import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt
from sklearn import tree

start_time = time.time()
log=open('classification_demo.txt','w') # test file

def loadData(fname):

    labels=[]
    reviews=[]
    f=open(fname)
    for line in f:
        rating,vote,review=line.strip().split('\t')
        reviews.append(review.lower())
        labels.append(rating.lower())
    f.close()
    return reviews,labels

question_train,labels_train=loadData('train.txt')
question_test,labels_test=loadData('test.txt')

#Build a counter based on the training dataset
counter = CountVectorizer(stop_words=stopwords.words('english'))
counter.fit(question_train)


#count the number of times each term appears in a document and transform each doc into a count v
counts_train = counter.transform(question_train)#transform the training data
counts_test = counter.transform(question_test)#transform the testing data

#    Classifiers Used:
#    1. KNN
#    2.Logistic Regression
#    3.DecisionTreeClassifier
#    4.RandomForestClassifier
#    5. Bayesian

KNN_classifier=KNeighborsClassifier()
LREG_classifier=LogisticRegression()
DT_classifier = DecisionTreeClassifier()
```

```
RF_classifier=RandomForestClassifier()
NB_classifier=MultinomialNB()


predictors=[('knn',KNN_classifier),('lreg',LREG_classifier),('dt',DT_classifier),('rf', RF_class
VT=VotingClassifier(predictors)


#==============================================================================
#build the parameter grid
KNN_grid = [{'n_neighbors': [1,3,5,7,9,11,13,15,17], 'weights':['uniform','distance']}]

#build a grid search to find the best parameters
gridsearchKNN = GridSearchCV(KNN_classifier, KNN_grid, cv=5)

#run the grid search
gridsearchKNN.fit(counts_train,labels_train)

#==============================================================================

#build the parameter grid
DT_grid = [{'max_depth': [3,4,5,6,7,8,9,10,11,12],'criterion':['gini','entropy']}]

#build a grid search to find the best parameters
gridsearchDT  = GridSearchCV(DT_classifier, DT_grid, cv=5)

#run the grid search
gridsearchDT.fit(counts_train,labels_train)

DT = tree.DecisionTreeClassifier()
treeClf = DT.fit(counts_train,labels_train)
tree.export_graphviz(treeClf, out_file='DTree.dot')
#==============================================================================

#build the parameter grid
LREG_grid = [ {'C':[0.5,1,1.5,2],'penalty':['l1','l2']}]

#build a grid search to find the best parameters
gridsearchLREG  = GridSearchCV(LREG_classifier, LREG_grid, cv=5)

#run the grid search
gridsearchLREG.fit(counts_train,labels_train)

#==============================================================================
#build the parameter grid
RF_grid = [{'n_estimators':[500,1000,1500,2000,2500,3000], 'criterion':['gini','entropy']}]

#build a grid search to find the best parameters
gridsearchRF = GridSearchCV(RF_classifier, RF_grid, cv=5)

#run the grid search
gridsearchRF.fit(counts_train,labels_train)


#==============================================================================
#build the parameter grid
```

```
NB_grid = [{'alpha':[0,1.0,5.0,10.0,20.0,30.0,40.0], 'fit_prior':[True,False]}]

#build a grid search to find the best parameters
gridsearchNB = GridSearchCV(NB_classifier, NB_grid, cv=5)

#run the grid search
gridsearchNB.fit(counts_train,labels_train)

#=============================================================================

VT.fit(counts_train,labels_train)

#use the VT classifier to predict
predicted=VT.predict(counts_test)

#print the accuracy
print (accuracy_score(predicted,labels_test))
log.write(str(accuracy_score(predicted,labels_test))+"\n");

algos=[];
acc = [];
m = 0;

for params, mean_score, scores in gridsearchKNN.grid_scores_:
    print (params, mean_score)
    log.write(str(params)+str(mean_score)+"\n")
    if mean_score>m:
        m = mean_score

acc.append(m)
m=0

for params, mean_score, scores in gridsearchDT.grid_scores_:
    print (params, mean_score)
    log.write(str(params)+str(mean_score)+"\n")
    if mean_score>m:
        m = mean_score

acc.append(m)
m=0

for params, mean_score, scores in gridsearchLREG.grid_scores_:
    print (params, mean_score)
    log.write(str(params)+str(mean_score)+"\n")
    if mean_score>m:
        m = mean_score

acc.append(m)
m=0

for params, mean_score, scores in gridsearchRF.grid_scores_:
    print (params, mean_score)
    log.write(str(params)+str(mean_score)+"\n")
    if mean_score>m:
        m = mean_score
```

```
acc.append(m)
m=0

for params, mean_score, scores in gridsearchNB.grid_scores_:
    print (params, mean_score)
    log.write(str(params)+str(mean_score)+"\n")
    if mean_score>m:
        m = mean_score

acc.append(m)
m=0

print(str(algos))
print(str(acc))

print("--- %s seconds ---" % round(time.time() - start_time, 2))

algos = ['KNN', 'DT', 'LREG', 'RF', 'NB']
y_pos = np.arange(len(algos))


plt.bar(y_pos, acc, align='center', alpha=0.5)
plt.xticks(y_pos, algos)
plt.ylabel('Accuracy')
plt.title('Algorithms v Accuracy')

plt.show()
```