```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 19 14:35:38 2017

@author: Harsh Kevadia
"""

from bs4 import BeautifulSoup
import re
import requests
import random

def run(url):

    fin=open('tags.txt','r') # input file
    fw=open('questions.txt','w') # output file
    log=open('log.txt','w') # log file
    test=open('test.txt','w') # test file
    train = open('train.txt','w') # training file

    c = 0 #just a counter for questions scraped
    c1 = 0
    c2 = 0

    vote_cutoff = -1 #vote threshold
    pageNum=42 # number of pages to collect
    test_percent = 0.1 #

    for line in fin: #for each tag
        html=None
        tag=line.lower().strip()
        print(tag)
        log.write('tag: ')
        log.write(tag+'\n')

        t =0
        t1 = 0
        t2 = 0

        for n in range(pageNum):
            if n == 0: continue
            pageLink=url+tag+'-interview-questions&n='+str(n) # make the page url
            arr1 = []

            for i in range(5): # try 5 times
                try:
                    #use the browser to access the url
                    response=requests.get(pageLink,headers = { 'User-Agent': 'Mozilla/5.0 (Windows
                    html=response.content # get the html
                    break # we got the file, break the loop
                except Exception as e:# threw an exception, the attempt to get the response failed
                    print ('failed attempt',i)
                    #time.sleep(2) # wait 2 secs


            if not html:continue # couldnt get the page, ignore
```

```
            soup = BeautifulSoup(html.decode('ascii', 'ignore'),'lxml') # parse the html

            questions=soup.findAll('li', {'class':re.compile('question')}) # get all the question c

            for question in questions:
                if question == questions[0]:
                    print('Page '+str(n))
                    log.write('Page '+str(n)+'\n')
                votes,text='NA','NA' # initialize votes and text
                votes = int(question.find('div', {'class':re.compile("votesNetQuestion")}).text)
                if votes > vote_cutoff :
                    text = question.find('p').text.replace("\r", " ").replace("\n", " ").replace("\
                    arr1.append(tag + "\t" + str(votes) + "\t" + text +'\n')
                    fw.write(tag + "\t" + str(votes) + "\t" + text +'\n')
                    c=c + 1
                    t = t + 1

            for i in range(int(len(arr1) * test_percent)):
                index = random.sample(range(len(arr1)),1)[0]
                test.write(arr1[index])
                arr1.remove(arr1[index])
                c1 = c1 + 1
                t1 = t1 + 1

            for i in range(len(arr1)):
                train.write(arr1[i])
                c2 = c2 + 1
                t2 = t2 + 1

        print('Questions Added to Tag:'+str(t))
        log.write('Questions Added to Tag:'+str(t)+'\n')
        print('Questions Added to Test:'+str(t1))
        log.write('Questions Added to Test:'+str(t1)+'\n')
        print('Questions Added to Train:'+str(t2))
        log.write('Questions Added to Train:'+str(t2)+'\n')
        print('Total Questions Added to Testing:'+str(c1))
        log.write('Total Questions Added to Testing:'+str(c1)+'\n')
        print('Total Questions Added to Training:'+str(c2))
        log.write('Total Questions Added to Training:'+str(c2)+'\n\n')

        print('Total Questions Scraped:'+str(c))
        log.write('Total Questions Scraped:'+str(c)+'\n')

    fin.close()
    fw.close()
    log.close()
    train.close()
    test.close()

if __name__=='__main__':
    url='https://www.careercup.com/page?pid='
    run(url)
```

```
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()
```

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 24 14:28:46 2017

@author: Harsh Kevadia
"""

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

def parse(file):
    fin=open(file)
    tags = []
    votes = []
    questions = []
    for line in fin: # for every line in the file (1 review per line)

        line=line.lower().strip()

        dataLine=line.split('\t')
        if(len(dataLine) == 3):
            if(int(dataLine[1]) > -1):
                for i in range(int(dataLine[1]) + 1):
                    tags.append(dataLine[0])
                    votes.append(dataLine[1])
                    questions.append(dataLine[2])
    fin.close()
    return tags,votes,questions
if __name__ == "__main__":
    train_tags,train_votes,train_questions = parse('train.txt')
    test_tags,test_votes,test_questions = parse('test.txt')

    vectorizer = TfidfVectorizer(stop_words='english')
    train = vectorizer.fit_transform(train_questions)
    test = vectorizer.transform(test_questions)

    clf = MLPClassifier(solver='lbgfs', alpha=1e-10, hidden_layer_sizes=(50,50), max_iter=100, ranc
    #, beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, learning_rate='constant', lec

    #clf =  SVC(gamma=2, C=1)
    #clf = MultinomialNB(alpha=0.1)
    #clf = GaussianNB()
    clf.fit(train,train_tags)
    pred=clf.predict(test)

    print (accuracy_score(pred,test_tags))
```

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 24 14:28:46 2017

@author: Harsh Kevadia
"""

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

def parse(file):
    fin=open(file)
    tags = []
    votes = []
    questions = []
    for line in fin: # for every line in the file (1 review per line)

        line=line.lower().strip()

        dataLine=line.split('\t')
        if(len(dataLine) == 3):
            if(int(dataLine[1]) > -1):
                for i in range(int(dataLine[1]) + 1):
                    tags.append(dataLine[0])
                    votes.append(dataLine[1])
                    questions.append(dataLine[2])
    fin.close()
    return tags,votes,questions
if __name__ == "__main__":
    train_tags,train_votes,train_questions = parse('train.txt')
    test_tags,test_votes,test_questions = parse('test.txt')

    vectorizer = TfidfVectorizer(stop_words='english')
    train = vectorizer.fit_transform(train_questions)
    test = vectorizer.transform(test_questions)

    clf = MultinomialNB(alpha=0.1)

    clf.fit(train,train_tags)
    pred=clf.predict(test)

    print (accuracy_score(pred,test_tags))
```

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 24 14:28:46 2017

@author: Harsh Kevadia
"""

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

def parse(file):
    fin=open(file)
    tags = []
    votes = []
    questions = []
    for line in fin: # for every line in the file (1 review per line)

        line=line.lower().strip()

        dataLine=line.split('\t')
        if(len(dataLine) == 3):
            if(int(dataLine[1]) > -1):
                for i in range(int(dataLine[1]) + 1):
                    tags.append(dataLine[0])
                    votes.append(dataLine[1])
                    questions.append(dataLine[2])
    fin.close()
    return tags,votes,questions
if __name__ == "__main__":
    train_tags,train_votes,train_questions = parse('train.txt')
    test_tags,test_votes,test_questions = parse('test.txt')

    vectorizer = TfidfVectorizer(stop_words='english')
    train = vectorizer.fit_transform(train_questions)
    test = vectorizer.transform(test_questions)

    clf =  SVC(gamma=2, C=1)
    #clf = MultinomialNB(alpha=0.1)
    #clf = GaussianNB()
    clf.fit(train,train_tags)
    pred=clf.predict(test)

    print (accuracy_score(pred,test_tags))
```

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 29 19:12:13 2017

@author: Harsh Kevadia
"""
import requests
import re
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt

def run(url):
    jobCounter = 0 #just a counter for questions scraped
    rcounter = 0
    mlCounter = 0
    bdCounter = 0
    pythonCounter = 0
    sasCounter = 0

    pageNum=19 # number of pages to collect

    for n in range(pageNum):
        if n == 0: continue
        pageLink=url + str(n) # make the page url\
        html = None
        print(pageLink)
        for i in range(5): # try 5 times
            try:
            #use the browser to access the url
                response = requests.get(pageLink,headers = { 'User-Agent': 'Mozilla/5.0 (Windows NT
                html = response.content # get the html
                #print(html)
                break # we got the file, break the loop
            except Exception as e:# threw an exception, the attempt to get the response failed
                print ('failed attempt Page URL',i)
                continue
                #time.sleep(2) # wait 2 secs

        if not html:
            continue
        soup = BeautifulSoup(html,'html.parser')
        jobs = soup.find_all('div', {'class':re.compile('jobTitle')})

        for job in jobs:
            for a in job.find_all('a', href=True):
                jobCounter = jobCounter + 1
                data = None
                for j in range(5):
                    try:
                        resp = requests.get(a['href'],headers = { 'User-Agent': 'Mozilla/5.0 (Windo
                        data = resp.content
                        break
                    except Exception as ex:
                        print('failed attempt JOB URL',i)
                        continue
                if not data:
                    continue
```

```
                data = str(data).strip('').lower()
                if 'python' in data:
                    pythonCounter = pythonCounter + 1
                if 'big data' in data:
                    bdCounter = bdCounter + 1
                if 'sas' in data:
                    sasFlag = False
                    if ' sas,' in data:
                        sasFlag = True
                    if ',sas ' in data:
                        sasFlag = True
                    if ' sas ' in data:
                        sasFlag = True
                    if sasFlag is True:
                        sasCounter = sasCounter + 1
                if 'r' in data:
                    rFlag = False
                    if ' r,' in data:
                        rFlag = True
                    if ',r ' in data:
                        rFlag = True
                    if ' r ' in data:
                        rFlag = True
                    if rFlag is True:
                        rcounter = rcounter + 1
                if 'machine learning' in data:
                    mlCounter = mlCounter + 1

    return jobCounter,pythonCounter,bdCounter,sasCounter,rcounter,mlCounter
if __name__=='__main__':

    resultFile=open('analysis.txt','w') # input file

    url='https://www.monster.com/jobs/search/?q=data-science&page='

    jobCounter,pythonCounter,bdCounter,sasCounter,rcounter,mlCounter = run(url)

    print('Total Jobs:' + str(jobCounter))
    print('Total Python Skill Jobs:' + str(pythonCounter) + 'Python Percentage: ' + str(pythonCount
    print('Total Big Data Skill Jobs:' + str(bdCounter) + 'Big Data Percentage: ' + str(bdCounter/j
    print('Total SAS Skill Jobs:' + str(sasCounter) + 'SAS Percentage: ' + str(sasCounter/jobCounte
    print('Total R Skill Jobs:' + str(rcounter) + 'R Percentage: ' + str(rcounter/jobCounter*100) +
    print('Total Machine Learning Skill Jobs:' + str(mlCounter) + 'Machine Learning Percentage: ' +

    resultFile.write('Total Jobs:' + str(jobCounter) + "\n")
    resultFile.write('Total Python Skill Jobs:' + str(pythonCounter) + 'Python Percentage: ' + str(
    resultFile.write('Total Big Data Skill Jobs:' + str(bdCounter) + 'Big Data Percentage: ' + str(
    resultFile.write('Total SAS Skill Jobs:' + str(sasCounter) + 'SAS Percentage: ' + str(sasCounte
    resultFile.write('Total R Skill Jobs:' + str(rcounter) + 'R Percentage: ' + str(rcounter/jobCou
    resultFile.write('Total Machine Learning Skill Jobs:' + str(mlCounter) + 'Machine Learning Perc

    labels = 'Python', 'R', 'Big Data', 'SAS', 'Machine Learning'
    data = [pythonCounter, rcounter, bdCounter, sasCounter, mlCounter]
    colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'lightpink']
    explode = (0, 0, 0, 0, 0)  # explode 1st slice

    # Plot
```

```
plt.pie(data, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, st

plt.axis('equal')
plt.show()

resultFile.close()
```