INLS Homework 3

Sarah Ganci

due: 11/26/2018

Read in Data

In [1]:
```
import pandas as pd
import numpy as np
import scipy
from sklearn import preprocessing
import csv
```

Training Data

In [2]:
```
df_annotators = pd.read_csv("hw3_data/hw3_data/music.train.annotators1.csv", e
ncoding='ISO-8859-1')
```

In [3]:
```
df_annotators.head(1)
```

Out[3]:

	text	a1	a2	a3	a4	a5	a6	a7	a8
0	oh! boy! how things have change..years ago i ...	positive	positive	positive	positive	positive	positive	positive	negative

In [4]:
```
df_annotators.shape
```

Out[4]: (997, 9)

Test Data

In [5]:
```
from sklearn.naive_bayes import MultinomialNB
```

In [6]:
```
df_test=pd.read_csv("hw3_data/hw3_data/music.test.csv",encoding='ISO-8859-1')
```

In [7]:
```
df_test.head(2)
```

Out[7]:

	text	class
0	joan. you don't love me. because i'm a guy. b...	positive
1	so far so good...so what! (in it's origional ...	negative

In [8]:
```
class_y = ['negative','positive']
```

In [9]:
```
le = preprocessing.LabelEncoder()
le.fit(class_y)
```

Out[9]:
```
LabelEncoder()
```

In [10]:
```
test_y= le.transform(df_test['class'])
```

Methods of Annotation Combination

Method 1: Score By Majority

In [11]:
```
#name: majority_wins
#in: row from dataframe object (expects a list)
#out: returns 1 if pos is >=neg, returns 0 if neg > pos
def majority_wins(row):
    neg= row[1:9].count("negative")
    pos=row[1:9].count("positive")
   # print("pos: "+str(pos)+ " neg: "+str(neg))
    return (0,1)[pos >= neg]
```

In [12]:
```
#Test data
df_annotators.as_matrix()[1].tolist()[1:9]
```

Out[12]:
```
['positive',
 'negative',
 'negative',
 'positive',
 'negative',
 'positive',
 'negative',
 'negative']
```

In [13]:
```
#Test function
majority_wins(df_annotators.as_matrix()[1].tolist())
```

Out[13]:
```
0
```

In [14]:
```
def majority_wins_convert(df):
    out=[]
    matrix=df.as_matrix()
    for i in range(len(matrix)):
        out.append(majority_wins(matrix[i].tolist()))
    return out
```

In [15]:
```
method1_labels= majority_wins_convert(df_annotators)
```

Method 2: Instance Weighting

In [16]:
```
#name: confidence
#in: df row (expected list from df_annotators)
#out: a tuple with 1 for pos 0 for neg and the score of how confident the scor
e is
def confidence(row):
    neg= row[1:9].count("negative")
    pos=row[1:9].count("positive")
   # if neg< pos, then return 0 and the number by how much more neg than pos
    #if pos>=neg, then return 1 and the number by how much more pos than neg
    return ([0,neg-pos],[1,pos-neg])[pos >= neg]
```

In [17]:
```
confidence(df_annotators.as_matrix()[1].tolist())
```

Out[17]:
```
[0, 2]
```

In [18]:
```
#in: dataframe (expected df_annotators)
#out: new dataframe object with weighted rows (duplicates of rows) and labeled
 columns
def confidence_convert(df):
    #out x will hold the text data (with duplicates)
    out_x=[]

    #out y will hold the labels for each text data (also duplicates)
    out_y=[]

    #text_df stores the df text column for ease of adding
    text=df['text']

    #matrix holds the df in matrix form to be passed
    matrix=df.as_matrix()

    for i in range(len(text)):
        con=confidence(matrix[i].tolist())
        for j in range(con[1]):
            out_x.append(matrix[i][0])
            out_y.append(con[0])

    return [out_y,out_x]
```

In [19]:
```
method2_labels, method2_x=confidence_convert(df_annotators)
```

In [20]:
```
print("size of labels: "+ str(len(method2_labels)))
print("size of x: "+ str(len(method2_x)))
```

```
size of labels: 3808
size of x: 3808
```

Method 3: Best Annotators

In [21]:
```
from sklearn.model_selection import cross_val_score
```

In [22]:
```
def cross_val(classifier, x, y, cv):
    return np.mean(cross_val_score(classifier, x, y, cv=cv))
```

In [23]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
```

Cross Validate Score for Each Annotator

We can test the consistancy of each annotator by using their labels to make predictions. We can use crossvalidation to calculate the cross validation score. Annotators with higher scores will are more consistant in their ratings. Hypothetically, if we use the more consistant annotators, then the overall accuracy of the model should be better.

In [24]:
```
#make vectorizor
tf_m3 = TfidfVectorizer(min_df=1,stop_words='english',max_features=2000)
#extract features from train x
x_tfidf_m3=tf_m3.fit_transform(df_annotators['text']).toarray()
```

In [25]:
```
annotators=[]
for i in range(1,9):
    mnb1= MultinomialNB(alpha=1)
    annotators.append([i, cross_val(mnb1, x_tfidf_m3, df_annotators["a"+str(i
)], 10)])
```

In [26]:
```
sorted_annotators=sorted(annotators, key= lambda tup: tup[1])
for i in range(len(sorted_annotators)):
    print(sorted_annotators[i])
```

```
[4, 0.52251515151515149]
[6, 0.53556565656565658]
[5, 0.63298289012574727]
[3, 0.70008260826082602]
[1, 0.73621662166216617]
[2, 0.76528882888288818]
[7, 0.81554695469546945]
[8, 0.85156277056277063]
```

The annotators in order of their consistancy are: 8, 7, 2, 1, 3, 5, 6, 4

Methods now allow for selection of annotators

In [27]:
```
#name: confidence
#in: df row (expected list from df_annotators), column numbers in an array
#out: a tuple with 1 for pos 0 for neg and the score of how confident the scor
e is
def confidence_tators(row, tators):
    scores=[]
    for i in range(len(tators)):
        scores.append(row[tators[i]])
       # print(row[tators[i]])
    neg= scores.count("negative")
    pos=scores.count("positive")
   # if neg< pos, then return 0 and the number by how much more neg than pos
    #if pos>=neg, then return 1 and the number by how much more pos than neg
    return ([0,neg-pos],[1,pos-neg])[pos >= neg]


def majority_wins_tators_convert(df, tators):
    out=[]
    matrix=df.as_matrix()
    for i in range(len(matrix)):
        out.append(confidence_tators(matrix[i].tolist(),tators)[0])
    return out
```

In [28]:
```
#majority_wins_tators_convert(df_annotators, [8])
```

Test majority wins for all diff combos of top annotators

In [29]:
```
annotator_combo=[]
test_tators=[[8],[8,7], [8, 7, 2], [8, 7, 2, 1], [8, 7, 2, 1, 3], [8, 7, 2, 1,
 3, 5], [8, 7, 2, 1, 3, 5, 6], [8, 7, 2, 1, 3, 5, 6, 4]]
for i in range(len(test_tators)):
    mnb1= MultinomialNB(alpha=1)
   # print(test_tators[i])
    y_labels=majority_wins_tators_convert(df_annotators, test_tators[i])
   # print(y_labels)
    cv_score=cross_val(mnb1, x_tfidf_m3, y_labels, 10)
    annotator_combo.append([test_tators[i], cv_score])
#print(annotator_combo)
```

In [30]:
```
sorted_annotator_combo=sorted(annotator_combo, key= lambda tup: tup[1])
for i in range(len(sorted_annotator_combo)):
    print(sorted_annotator_combo[i])
```

```
[[8, 7, 2, 1, 3, 5, 6, 4], 0.73029828901257465]
[[8, 7, 2, 1, 3, 5, 6], 0.75024242424242427]
[[8, 7, 2, 1, 3], 0.75630818387961241]
[[8, 7, 2], 0.76030313031303132]
[[8, 7, 2, 1], 0.77134373437343728]
[[8, 7, 2, 1, 3, 5], 0.77230853085308537]
[[8, 7], 0.78637373737373739]
[[8], 0.85156277056277063]
```

The best combination of annotators was just annotator 8 (the top 1 annotator)

In [31]:
```
method3_labels_top1= majority_wins_tators_convert(df_annotators, [8])
```

In [32]:
```
method3_labels_top2=majority_wins_tators_convert(df_annotators, [8, 7])
```

In [33]:
```
method3_labels_top3=majority_wins_tators_convert(df_annotators, [8, 7, 2])
```

In [34]:
```
method3_labels_top4=majority_wins_tators_convert(df_annotators, [8, 7, 2, 1])
```

In [35]:
```
method3_labels_top5=majority_wins_tators_convert(df_annotators, [8, 7, 2, 1, 3
])
```

In [36]:
```
method3_labels_top6=majority_wins_tators_convert(df_annotators, [8, 7, 2, 1, 3
, 5 ])
```

In [37]:
```
method3_labels_top7=majority_wins_tators_convert(df_annotators, [8, 7, 2, 1, 3
, 5, 6])
```

In [38]:
```
method3_labels_top8=majority_wins_tators_convert(df_annotators, [8, 7, 2, 1, 3
, 5, 6, 4])
```

Evaluation

Confusion Matrix Score

In [39]:
```
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

In [40]:
```
#in: dataframe of x features, y labels, test_x tfidf vector, test_y labels
#out: confusion matrix
def model_predict_print(train_x, train_y, test_x, test_y):
    #make model
    mnb = MultinomialNB(alpha=1.0)

    #make vectorizor
    tf = TfidfVectorizer(min_df=1,stop_words='english',max_features=2000)

    #extract features from train x
    x_tfidf=tf.fit_transform(train_x).toarray()

    #fit mnb model based on the train x text and train y labels
    mnb.fit(x_tfidf,train_y)

    #create x_test_tfidf by transforming test_x using tf
    x_test_tfidf=tf.transform(test_x).toarray()

    #store predictions from x_test tfidf array in predictions
    predictions=mnb.predict(x_test_tfidf)

    #calculate confusion matrix using method
    print("Confusion Matrix:")

    print(confusion_matrix(test_y, predictions ))
    #calculate accuracy using accuracy_score method

    print("Accuracy: ")
    print(accuracy_score(test_y, predictions))
```

Method 1: Score By Majority Evaluation

In [41]:
```
model_predict_print(df_annotators['text'],method1_labels,df_test['text'],test_y)
```

```
Confusion Matrix:
[[360 121]
 [126 393]]
Accuracy:
0.753
```

Method 2: Instance Weighting Evaluation

In [42]:
```
model_predict_print(method2_x,method2_labels,df_test['text'],test_y)
```

```
Confusion Matrix:
[[427  54]
 [247 272]]
Accuracy:
0.699
```

Method 3: Best Annotators

Top 1 Annotators

In [43]:
```
model_predict_print(df_annotators['text'],method3_labels_top1,df_test['text'],
test_y)
```

```
Confusion Matrix:
[[481   0]
 [518   1]]
Accuracy:
0.482
```

Top 2 Annotators

In [44]:
```
model_predict_print(df_annotators['text'],method3_labels_top2,df_test['text'],
test_y)
```

```
Confusion Matrix:
[[351 130]
 [122 397]]
Accuracy:
0.748
```

Top 3 Annotators

In [45]:
```
model_predict_print(df_annotators['text'],method3_labels_top3,df_test['text'],
test_y)
```

```
Confusion Matrix:
[[480   1]
 [518   1]]
Accuracy:
0.481
```

Top 4 Annotators

In [46]:
```
model_predict_print(df_annotators['text'],method3_labels_top4,df_test['text'],
test_y)
```

```
Confusion Matrix:
[[345 136]
 [118 401]]
Accuracy:
0.746
```

Top 5 Annotators

In [47]:
```
model_predict_print(df_annotators['text'],method3_labels_top5,df_test['text'],
test_y)
```

```
Confusion Matrix:
[[440  41]
 [248 271]]
Accuracy:
0.711
```

Top 6 Annotators

In [48]:
```
model_predict_print(df_annotators['text'],method3_labels_top6,df_test['text'],
test_y)
```

```
Confusion Matrix:
[[366 115]
 [129 390]]
Accuracy:
0.756
```

Top 7 Annotators

In [49]:
```
model_predict_print(df_annotators['text'],method3_labels_top7,df_test['text'],
test_y)
```

```
Confusion Matrix:
[[434  47]
 [230 289]]
Accuracy:
0.723
```

Top 8 Annotators (All Annotators)

In [50]:
```
model_predict_print(df_annotators['text'],method3_labels_top8,df_test['text'],
test_y)
```

```
Confusion Matrix:
[[360 121]
 [126 393]]
Accuracy:
0.753
```

Comparision

Method 3-6 (Majority Wins with Top 6 Annotators) achieved the highest accuracy

	Method 3-1	**Method 3-2**	**Method 3-3**	**Method 3-4**	**Method 3-5**	**Method 3-6**	**Method 3-7**	**Method 3-8**
Accuracy	0.482	0.748	0.481	0.746	0.711	**0.756**	0.723	0.753

	Method 1	**Method 2**	**Method 3-6**
Accuracy	0.753	0.699	**0.756**