HomeWork_1_CSE517

1 Text Classification – Eisenstein 4.6 (p. 89)

After you run tar -xzf A1.tgz, in the directory review polarity, you will find a dataset of positively and negatively classified reviews that was used by Pang and Lee [2], a seminal paper about sentiment classification. Consult the readme file for more information. Hold out a randomly selected 400 reviews as a test set.

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from math import sqrt
```

In this step below the text files from the pos folder are being read and has been transformed into dataframe (df_pos). This datapframe contains two columns namely "Reviews" and "Sentiments (with values 'Positive')".

In [39]:
```
# Import Module
import os

# Folder Path
path = r'/Users/nehakardam/Documents/UWclasses /CSE NLP/A1/A1_data/revi

# Change the directory
os.chdir(path)

# Read text File
def read_text_file(file_path):
    with open(file_path, 'r') as f:
        return f.read().splitlines()

# iterate through all file
collection_of_data_pos = []
for file in os.listdir():
    # Check whether file is in text format or not
    if file.endswith(".txt"):
        file_path = f"{path}/{file}"
        # call read text file function
        collection_of_data_pos = collection_of_data_pos + read_text_fil
```

In [40]:
```
df_pos = pd.DataFrame(collection_of_data_pos, columns = ['Reviews'])
df_pos['Sentiments'] = 'Positive'
```

In this step below the text files from the neg folder are being read and has been transformed into dataframe (df_pos). This datapframe contains two columns namely "Reviews" and "Sentiments (with values 'Negative')".

In [41]:
```
# Folder Path
path = r'/Users/nehakardam/Documents/UWclasses /CSE NLP/A1/A1_data/revi

# Change the directory
os.chdir(path)

# iterate through all file
collection_of_data_neg = []
for file in os.listdir():
    # Check whether file is in text format or not
    if file.endswith(".txt"):
        file_path = f"{path}/{file}"
        # call read text file function
        collection_of_data_neg = collection_of_data_neg + read_text_fil
```

In [42]:
```
df_neg = pd.DataFrame(collection_of_data_neg, columns = ['Reviews'])
df_neg['Sentiments'] = 'Negative'
```

In [43]:
```
#both the dataframe of positive and negative reviews are concatinated t
df_pos_neg = [df_pos, df_neg]
Collection_of_data = pd.concat(df_pos_neg)
#Shuffle the columns of the data
All_data= Collection_of_data.sample(frac=1).reset_index(drop=True)
All_data
```

Out[43]:

	Reviews	Sentiments
0	whew . okay .	Negative
1	warning : this review contains some spoilers f...	Positive
2	its nice to see a pg rated movie out at christ...	Positive
3	he gives each one a different series of cranky...	Negative
4	but like everything else , tarantino takes thi...	Positive
...	...	...
64715	everything about this movie was just completel...	Negative
64716	basically , he says , it boils down to a lack ...	Negative
64717	y'know , that just sends chills down my spine .	Negative
64718	bryan singer's " the usual suspects " for one ...	Negative
64719	i knew it was going to be a bad print with onl...	Negative

64720 rows × 2 columns

In [66]:
```
All_data.to_csv('/Users/nehakardam/Documents/UWclasses /CSE NLP/A1/A1_d
lex_neg = pd.read_csv('/Users/nehakardam/Documents/UWclasses /CSE NLP/A
lex_pos = pd.read_csv('/Users/nehakardam/Documents/UWclasses /CSE NLP/A
```

In [67]:
```
dict_neg_pos = {}
for index, value in lex_neg[0].items():
    dict_neg_pos[value] = "negative"
for index, value in lex_pos[0].items():
    dict_neg_pos[value] = "positive"
```

In [216]:
```
import re
def tokenizer(theText):
    theTokens = re.findall(r'\b\w[\w-]*\b', theText.lower())
    return theTokens

reviews_tokenized = []
reviews = All_data['Reviews']

for index, value in reviews.items():
    reviews_tokenized.append(tokenizer(value))
All_data['Reviews_Tokenized'] = reviews_tokenized

```

In [69]:
```
#Creating test data set
Testset_data = All_data.sample(n=400, random_state=1)
```

Part A: Sentiment lexicon-based classifier.

Sentiment lexicon-based classifier. Create a classifier using a sentiment lexicon. A lexicon from Hu and Liu [1] is provided in the directory opinion lexicon English, but you are welcome to find and use (with attribution, of course) another. Tokenize the data, and classify each document as positive if and only if it has more positive sentiment words than negative sentiment words. Compute and report the accuracy and F1 score (on detecting positive reviews) on the test set, using this lexicon-based classifier.

In [210]:
```
def result_of_sentiment_analysis(review):
    pos_count = 0
    neg_count = 0
    for token in review:
        if token in dict_neg_pos:
            s = dict_neg_pos.get(token)
            if s == "positive":
                pos_count = pos_count + 1
            else:
                neg_count = neg_count + 1
    if pos_count >= neg_count:
        return "Positive"
    else:
        return "Negative"

sentiments_y = []
reviews_tokens = Testset_data['Reviews_Tokenized']
for review in reviews_tokens:
    sentiments_y.append(result_of_sentiment_analysis(review))
```

In [211]:
```
Testset_data['Sentiments_Y'] = sentiments_y
Testset_data
```

Out[211]:

	Reviews	Sentiments	Reviews_Tokenized	Sentiments_Y
54756	you're so . . .	Negative	[you, re, so]	Positive
43964	no one would deny that spillane's writing has ...	Negative	[no, one, would, deny, that, spillane, s, writ...	Negative
21339	i can ? t recall seeing a film recently that w...	Positive	[i, can, t, recall, seeing, a, film, recently,...	Positive
56309	the times are changing , though , and the gear...	Positive	[the, times, are, changing, though, and, the, ...	Positive
54493	i certainly wish that was a new trend in holly...	Negative	[i, certainly, wish, that, was, a, new, trend,...	Positive
...	...	...	...	...
9945	details essential to the story are so improbab...	Negative	[details, essential, to, the, story, are, so, ...	Negative
62137	as many great movies that got cult status thro...	Positive	[as, many, great, movies, that, got, cult, sta...	Positive
10376	if we encourage terrorism abroad (from bening...	Positive	[if, we, encourage, terrorism, abroad, from, b...	Negative
50983	alas , the times are a-changin' .	Positive	[alas, the, times, are, a-changin]	Positive
6022	the chase is also shot and edited in that " je...	Positive	[the, chase, is, also, shot, and, edited, in, ...	Positive

400 rows × 4 columns

Accuracy and F1 Score calculation

In [72]:
```
actual = Testset_data['Sentiments']
predicted = Testset_data['Sentiments_Y']

tp = np.sum((actual=='Positive') & (predicted=='Positive'))
fp = np.sum((actual!='Positive') & (predicted=='Positive'))
tn = np.sum((actual=='Negative') & (predicted=='Negative'))
fn = np.sum((actual!='Negative') & (predicted=='Negative'))

accuracy = (tp+tn)/(tp+tn+fp+fn)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1 = 2 * (precision * recall) / (precision + recall)
```

In [73]:
```
print (accuracy, precision, recall, f1)
```

```
0.5375 0.4854014598540146 0.751412429378531 0.5898004434589801
```

Part B: Logistic regression classifier.

Train a (binary) logistic regression classifier on your training set using features of your own choosing, and report its accuracy and F1 score (as above) on the test set. In your write-up, describe the features you have chose and explain the reasoning behind your choice. Do not use pretrained word vectors or any features implemented or constructed by anyone else. Do not use an existing implementation of logistic regression, stochastic gradient descent, or automatic differentiation.

In [364]:
```
Training_data = All_data.copy()
Testset_data = All_data.sample(n=4000, random_state=1)
Training_data.drop(Testset_data.index)
Training_data = Training_data.sample(n=20000, random_state=1)
```

Bag of word Model

In [365]:
```
word_dict = {}

def add_all_workds_to_dict(words):
    for word in words:
        if word in word_dict:
            word_dict[word] = word_dict.get(word) + 1
        else:
            word_dict[word] = 1

for words in Training_data["Reviews_Tokenized"]:
    add_all_workds_to_dict(words)
```

In [366]:
```
frequent_word_set = set()
for key, value in word_dict.items():
    if value > 10:
        frequent_word_set.add(key)
```

In [367]:
```
len(frequent_word_set)
```

Out[367]: 3654

In [368]:
```
Training_data
```

Out[368]:

	Reviews	Sentiments	Reviews_Tokenized
54756	you're so . . .	Negative	[you, re, so]
43964	no one would deny that spillane's writing has ...	Negative	[no, one, would, deny, that, spillane, s, writ...
21339	i can ? t recall seeing a film recently that w...	Positive	[i, can, t, recall, seeing, a, film, recently,...
56309	the times are changing , though , and the gear...	Positive	[the, times, are, changing, though, and, the, ...
54493	i certainly wish that was a new trend in holly...	Negative	[i, certainly, wish, that, was, a, new, trend,...
...	...	...	...
44239	in pay it forward leder continues in this vein...	Negative	[in, pay, it, forward, leder, continues, in, t...
29593	the humor especially works well in this case ,...	Positive	[the, humor, especially, works, well, in, this...
22651	here's a film that actually gives away most of...	Positive	[here, s, a, film, that, actually, gives, away...
7397	a bad guy that's not really bad .	Positive	[a, bad, guy, that, s, not, really, bad]
10320	this vacuous picture throws in a standard down...	Negative	[this, vacuous, picture, throws, in, a, standa...

20000 rows × 3 columns

In [369]:
```
Training_data_with_features = pd.DataFrame(columns = frequent_word_set)
for review in Training_data["Reviews_Tokenized"]:
    i = len(Training_data_with_features)
    Training_data_with_features.loc[i] = 0
    for word in review:
        if word in frequent_word_set:
            count = Training_data_with_features.loc[i, word]
            Training_data_with_features.loc[i, word] = count + 1
train_X = Training_data_with_features.astype(float)
```

In [370]:
```
train_Y = []
for sentiment in Training_data["Sentiments"]:
    if sentiment == "Positive":
        train_Y.append(1)
    else:
        train_Y.append(0)
```

Logistic Regression Model Classifier

In [371]:
```
class Logistic_Regression() :
    def __init__(self, learning_rate=0.001, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for i in range(self.n_iters):
            model_linear = np.dot(X, self.weights) + self.bias
            predict_y = self._sigmoid(model_linear)
            dw = (1 / n_samples) * np.dot(X.T, (predict_y - y))
            db = (1 / n_samples) * np.sum(predict_y - y)
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        model_linear = np.dot(X, self.weights) + self.bias
        predict_y = self._sigmoid(model_linear)
        predict_y_cls = [1 if i > 0.5 else 0 for i in predict_y]
        return np.array(predict_y_cls)

    def _sigmoid(self, x):
        return 1 / (1 + np.exp(-x))
```

In [372]:
```
regression = Logistic_Regression()
regression.fit(train_X, train_Y)
```

In [373]:
```
Testset_data_with_features = pd.DataFrame(columns = frequent_word_set)
for review in Testset_data["Reviews_Tokenized"]:
    i = len(Testset_data_with_features)
    Testset_data_with_features.loc[i] = 0
    for word in review:
        if word in frequent_word_set:
            count = Testset_data_with_features.loc[i, word]
            Testset_data_with_features.loc[i, word] = count + 1
test_X = Testset_data_with_features.astype(float)
```

In [374]:
```
test_Y = []
for sentiment in Testset_data["Sentiments"]:
    if sentiment == "Positive":
        test_Y.append(1)
    else:
        test_Y.append(0)
```

In [375]:
```
predictions = regression.predict(test_X)
type(predictions)
```

Out[375]:
```
numpy.ndarray
```

In [376]:
```
actual = np.array(test_Y)
predicted = predictions

tp = np.sum((actual==1) & (predicted==1))
fp = np.sum((actual!=1) & (predicted==1))
tn = np.sum((actual==0) & (predicted==0))
fn = np.sum((actual!=0) & (predicted==0))

accuracy = (tp+tn)/(tp+tn+fp+fn)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1 = 2 * (precision * recall) / (precision + recall)
```

In [377]:
```
print (accuracy, precision, recall, f1)
```

```
0.53325 0.5237562884292901 0.9199803632793323 0.6674977738201247
```

Breaking good.

Case 1: Positive by human English speakers, and your lexicon classifier predicts it as positive, whereas your logistic regression classifier predicts it as negative.

In [340]:
```
text1 = tokenizer("I've recently noticed something I had not even consi
result_of_sentiment_analysis(text1)
```

Out[340]: 'Positive'

In [341]:
```
breaking_bad_text = text1
breaking_bad_text_with_features = pd.DataFrame(columns = frequent_word_
breaking_bad_text_with_features.loc[0] = 0
for word in breaking_bad_text:
    if word in frequent_word_set:
        count = breaking_bad_text_with_features.loc[0, word]
        breaking_bad_text_with_features.loc[0, word] = count + 1
breaking_bad_text_X = breaking_bad_text_with_features.astype(float)

regression.predict(breaking_bad_text_X) # 0 means negative and 1 means
```

Out[341]: array([0])

Case 2: Human positive and your lexicon classifier predicts it as negative, whereas your logistic regression classifier predicts it as positive.

In [380]:
```
text2 = tokenizer("You can bet that anybody who has ever seen Karan's m
result_of_sentiment_analysis(text2)
```

Out[380]: 'Negative'

In [381]:
```
breaking_bad_text = text2
breaking_bad_text_with_features = pd.DataFrame(columns = frequent_word_
breaking_bad_text_with_features.loc[0] = 0
for word in breaking_bad_text:
    if word in frequent_word_set:
        count = breaking_bad_text_with_features.loc[0, word]
        breaking_bad_text_with_features.loc[0, word] = count + 1
breaking_bad_text_X = breaking_bad_text_with_features.astype(float)

regression.predict(breaking_bad_text_X) # 0 means negative and 1 means
```

Out[381]: array([1])

Case 3: Both of your classifiers predict it as negative.

In [409]:
```
text3 = tokenizer("They're all unhappy or marginally so about noodles i
result_of_sentiment_analysis(text3)
```

Out[409]: 'Negative'

In [410]:
```
breaking_bad_text = text3
breaking_bad_text_with_features = pd.DataFrame(columns = frequent_word_
breaking_bad_text_with_features.loc[0] = 0
for word in breaking_bad_text:
    if word in frequent_word_set:
        count = breaking_bad_text_with_features.loc[0, word]
        breaking_bad_text_with_features.loc[0, word] = count + 1
breaking_bad_text_X = breaking_bad_text_with_features.astype(float)

regression.predict(breaking_bad_text_X) # 0 means negative and 1 means
```

Out[410]: array([0])

In [ ]:
```

```