1. Test the TfidfVectorizer in place of CountVectorizer on the IMDB data. Do you see any difference in the classification results or the optimal C value?

In [1]:

```
import json
import random
with open("imdb_train.json") as f:
    data=json.load(f)
random.shuffle(data)
print("class label:", data[0]["class"])
print("text:",data[0]["text"])
```

```
class label: neg
text: The film starts to slowly when we got to the cinema we thought it looked
quite good but after about 5 mins we were all bored out of our minds and wonde
ring what kind of film we had come to see, i don't like this film and wouldn't
recommend it to anyone, the best part of the night was when the alarm and ligh
ts came back on because the project broke down because we thought we could all
go home. this has to be one of the worst films i have ever seen we were all bo
red out of or minds and most of the people in the cinema actually RAN out of t
he doors at the end because it was so rubbish. i am surprised that no one walk
ed out earlier than that. if you go and see it make sure you something to keep
you busy, better still Don't go and see it at all.
```

In [2]:

```
# We need to gather the texts and labels into separate lists
texts=[one_example["text"] for one_example in data]
labels=[one_example["class"] for one_example in data]
print("This many texts",len(texts))
print("This many labels",len(labels))
print()
for label,text in list(zip(labels,texts))[:20]:
    print(label,text[:50]+"...")
```

```
This many texts 25000
This many labels 25000

neg The film starts to slowly when we got to the cinem...
pos This is one of my all-time favorite films, and whi...
pos I used to watch this show when I was a little girl...
neg My dad is a fan of Columbo and I had always dislik...
pos I won't claim to be a fan of Ralph Bakshi, because...
neg If I had known this movie was filmed in the exaspe...
neg Im watching it now on pink (Serbia TV station) and...
neg - A newlywed couple move into the home of the husb...
pos This movie was excellent for the following reasons...
neg i am a big fan of karishma Kapoor and Govinda. I w...
neg The best Laurel and Hardy shorts are filled to the...
neg Not a `woman film' but film for the gang. One of t...
neg A dedicated fan to the TLK movies, with the first ...
neg THE CHOKE (aka AXE in the UK) is a slasher produce...
neg Wow... what would you do with $33m? Let me give yo...
neg I dunno what the hype around this is... This is re...
pos Broadway and film actor-turned-director John Cassa...
neg I have never seen one of these SciFi originals bef...
pos As a low budget enterprise in which the filmmakers...
pos \Bedknobs and Broomsticks\" is a magical adventure...
```

In [3]:

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split #split data

train_texts, dev_texts, train_labels, dev_labels=train_test_split(texts,label
tfidfvectorizer=TfidfVectorizer(max_features=100000,binary=True,ngram_range=(
feature_matrix_train=tfidfvectorizer.fit_transform(train_texts)
feature_matrix_dev=tfidfvectorizer.transform(dev_texts)
```

In [4]:

```python
print(feature_matrix_train.shape)
print(feature_matrix_dev.shape)
```

```
(20000, 68452)
(5000, 68452)
```

Optimal C value = 0.0005

In [5]:

```python
import sklearn.svm #train via classifier
classifier=sklearn.svm.LinearSVC(C=0.0005,verbose=1)
classifier.fit(feature_matrix_train, train_labels)
```

```
[LibLinear]
```

Out[5]:

```
LinearSVC(C=0.0005, verbose=1)
```

In [6]:

```
#Test data
print("DEV",classifier.score(feature_matrix_dev, dev_labels))
print("TRAIN",classifier.score(feature_matrix_train, train_labels))
```

```
DEV 0.8128
TRAIN 0.83585
```

In [10]:

```
import sklearn.metrics
predictions_dev=classifier.predict(feature_matrix_dev)
print(predictions_dev)
print(sklearn.metrics.confusion_matrix(dev_labels,predictions_dev))
print(sklearn.metrics.accuracy_score(dev_labels,predictions_dev))
```

```
['neg' 'pos' 'pos' ... 'neg' 'pos' 'neg']
[[1863  690]
 [ 246 2201]]
0.8128
```

In [8]:

```
import pickle

with open("saved_model.pickle","wb") as f:
    pickle.dump((classifier,tfidfvectorizer),f)
```

In [9]:

```
with open("saved_model.pickle","rb") as f:
    classifier_loaded,vectorizer_loaded=pickle.load(f)

feature_matrix_dev_loaded=vectorizer_loaded.transform(dev_texts)
print("DEV - loaded (should match the score above)",classifier_loaded.score(f
```

```
DEV - loaded (should match the score above) 0.8128
```

- C value = 0.0005 returns dev set with accuracy score of 83.54%
- the result from tfidfvectorizer is not as good as countvectorizer

Optimal C value = 0.05

In [7]:

```
import sklearn.svm #train via classifier
classifier2=sklearn.svm.LinearSVC(C=0.05,verbose=1)
classifier2.fit(feature_matrix_train, train_labels)
```

```
[LibLinear]
```

Out[7]:

```
LinearSVC(C=0.05, verbose=1)
```

In [8]:

```
#Test data
print("DEV",classifier2.score(feature_matrix_dev, dev_labels))
print("TRAIN",classifier2.score(feature_matrix_train, train_labels))
```

```
DEV 0.8934
TRAIN 0.922
```

C value = 0.05 returns dev set with accuracy score of 89.34%

Optimal C value = 0.5

In [9]:

```
import sklearn.svm #train via classifier
classifier3=sklearn.svm.LinearSVC(C=0.5,verbose=1)
classifier3.fit(feature_matrix_train, train_labels)
```

```
[LibLinear]
```

Out[9]:

```
LinearSVC(C=0.5, verbose=1)
```

In [10]:

```
#Test data
print("DEV",classifier3.score(feature_matrix_dev, dev_labels))
print("TRAIN",classifier3.score(feature_matrix_train, train_labels))
```

```
DEV 0.8944
TRAIN 0.98505
```

C value = 0.5 returns dev set with accuracy score of 89.44%

1. Test different lengths of n-grams in the CountVectorizer on the IMDB data. Do you see any difference in the classification results or the optimal C value?

N-grams = 1, C = 0.0005

In [11]:

```
import sklearn
from sklearn.feature_extraction.text import CountVectorizer
train_texts, dev_texts, train_labels, dev_labels=train_test_split(texts,label
vectorizer=CountVectorizer()
```

In [12]:

```
from sklearn.model_selection import train_test_split #split data

train_texts, dev_texts, train_labels, dev_labels=train_test_split(texts,label
vectorizer1=CountVectorizer(max_features=100000,binary=True,ngram_range=(1,1)
feature_matrix_train1=vectorizer1.fit_transform(train_texts)
feature_matrix_dev1=vectorizer1.transform(dev_texts)
```

In [13]:

```
print(feature_matrix_train1.shape)
print(feature_matrix_dev1.shape)
```

```
(20000, 68298)
(5000, 68298)
```

In [14]:

```
import sklearn.svm #train via classifier
classifier4=sklearn.svm.LinearSVC(C=0.0005,verbose=1)
classifier4.fit(feature_matrix_train1, train_labels)
```

```
[LibLinear]
```

Out[14]:

```
LinearSVC(C=0.0005, verbose=1)
```

In [15]:

```
#Test data
print("DEV",classifier4.score(feature_matrix_dev1, dev_labels))
print("TRAIN",classifier4.score(feature_matrix_train1, train_labels))
```

```
DEV 0.8662
TRAIN 0.8954
```

In [16]:

```
# Test C value and see the results
import sklearn.metrics
predictions_dev1=classifier4.predict(feature_matrix_dev1)
print(predictions_dev1)
print(sklearn.metrics.confusion_matrix(dev_labels,predictions_dev1))
print(sklearn.metrics.accuracy_score(dev_labels,predictions_dev1))
```

```
['pos' 'neg' 'pos' ... 'neg' 'neg' 'neg']
[[2163  358]
 [ 311 2168]]
0.8662
```

N-grams = 1, C = 0.005

In [17]:

```
import sklearn.svm #train via classifier
classifier4=sklearn.svm.LinearSVC(C=0.005,verbose=1)
classifier4.fit(feature_matrix_train1, train_labels)
```

```
[LibLinear]
```

Out[17]:

```
LinearSVC(C=0.005, verbose=1)
```

In [18]:

```
#Test data
print("DEV",classifier4.score(feature_matrix_dev1, dev_labels))
print("TRAIN",classifier4.score(feature_matrix_train1, train_labels))
```

```
DEV 0.8796
TRAIN 0.9566
```

In [19]:

```
# Test C value and see the results
import sklearn.metrics
predictions_dev1=classifier4.predict(feature_matrix_dev1)
print(predictions_dev1)
print(sklearn.metrics.confusion_matrix(dev_labels,predictions_dev1))
print(sklearn.metrics.accuracy_score(dev_labels,predictions_dev1))
```

```
['pos' 'neg' 'pos' ... 'neg' 'neg' 'neg']
[[2205  316]
 [ 286 2193]]
0.8796
```

N-gram = 2, C = 0.0005

In [20]:

```
from sklearn.model_selection import train_test_split #split data

train_texts, dev_texts, train_labels, dev_labels=train_test_split(texts,label
vectorizer2=CountVectorizer(max_features=None,binary=True,ngram_range=(2,2))
feature_matrix_train2=vectorizer2.fit_transform(train_texts)
feature_matrix_dev2=vectorizer2.transform(dev_texts)
```

In [21]:

```
print(feature_matrix_train2.shape)
print(feature_matrix_dev2.shape)
```

```
(20000, 1224196)
(5000, 1224196)
```

In [22]:

```
import sklearn.svm #train via classifier
classifier5=sklearn.svm.LinearSVC(C=0.0005,verbose=1)
classifier5.fit(feature_matrix_train2, train_labels)
```

```
[LibLinear]
```

Out[22]:

```
LinearSVC(C=0.0005, verbose=1)
```

In [23]:

```
#Test data
print("DEV",classifier5.score(feature_matrix_dev2, dev_labels))
print("TRAIN",classifier5.score(feature_matrix_train2, train_labels))
```

```
DEV 0.855
TRAIN 0.9384
```

In [24]:

```
# Test C value and see the results
import sklearn.metrics
predictions_dev2=classifier5.predict(feature_matrix_dev2)
print(predictions_dev2)
print(sklearn.metrics.confusion_matrix(dev_labels,predictions_dev2))
print(sklearn.metrics.accuracy_score(dev_labels,predictions_dev2))
```

```
['neg' 'neg' 'pos' ... 'pos' 'pos' 'pos']
[[2079  424]
 [ 301 2196]]
0.855
```

N-grams = 2 Countvectorizer, C = 0.005

In [25]:

```
import sklearn.svm #train via classifier
classifier5=sklearn.svm.LinearSVC(C=0.005,verbose=1)
classifier5.fit(feature_matrix_train2, train_labels)
```

```
[LibLinear]
```

Out[25]:

```
LinearSVC(C=0.005, verbose=1)
```

In [26]:

```
#Test data
print("DEV",classifier5.score(feature_matrix_dev2, dev_labels))
print("TRAIN",classifier5.score(feature_matrix_train2, train_labels))
```

```
DEV 0.877
TRAIN 0.99865
```

In [27]:

```
# Test C value and see the results
import sklearn.metrics
predictions_dev2=classifier5.predict(feature_matrix_dev2)
print(predictions_dev2)
print(sklearn.metrics.confusion_matrix(dev_labels,predictions_dev2))
print(sklearn.metrics.accuracy_score(dev_labels,predictions_dev2))
```

```
['neg' 'neg' 'neg' ... 'pos' 'pos' 'pos']
[[2153  350]
 [ 265 2232]]
0.877
```

N-gram = 3, C = 0.0005

In [28]:

```
from sklearn.model_selection import train_test_split #split data

train_texts, dev_texts, train_labels, dev_labels=train_test_split(texts,label
vectorizer3=CountVectorizer(max_features=None,binary=True,ngram_range=(3,3))
feature_matrix_train3=vectorizer3.fit_transform(train_texts)
feature_matrix_dev3=vectorizer3.transform(dev_texts)
```

In [29]:

```
print(feature_matrix_train3.shape)
print(feature_matrix_dev3.shape)
```

```
(20000, 3013093)
(5000, 3013093)
```

In [30]:

```
import sklearn.svm #train via classifier
classifier6=sklearn.svm.LinearSVC(C=0.0005,verbose=1)
classifier6.fit(feature_matrix_train3, train_labels)
```

```
[LibLinear]
```

Out[30]:

```
LinearSVC(C=0.0005, verbose=1)
```

In [31]:

```
#Test data
print("DEV",classifier6.score(feature_matrix_dev3, dev_labels))
print("TRAIN",classifier6.score(feature_matrix_train3, train_labels))
```

```
DEV 0.7804
TRAIN 0.97205
```

In [32]:

```
# Test C value and see the results
import sklearn.metrics
predictions_dev3=classifier6.predict(feature_matrix_dev3)
print(predictions_dev3)
print(sklearn.metrics.confusion_matrix(dev_labels,predictions_dev3))
print(sklearn.metrics.accuracy_score(dev_labels,predictions_dev3))
```

```
['neg' 'neg' 'pos' ... 'pos' 'pos' 'pos']
[[1877  600]
 [ 498 2025]]
0.7804
```

N-gram = 3, C = 0.005

In [33]:

```
import sklearn.svm #train via classifier
classifier6=sklearn.svm.LinearSVC(C=0.005,verbose=1)
classifier6.fit(feature_matrix_train3, train_labels)
```

```
[LibLinear]
```

Out[33]:

```
LinearSVC(C=0.005, verbose=1)
```

In [34]:

```
#Test data
print("DEV",classifier6.score(feature_matrix_dev3, dev_labels))
print("TRAIN",classifier6.score(feature_matrix_train3, train_labels))
```

```
DEV 0.817
TRAIN 0.99975
```

In [35]:

```
# Test C value and see the results
import sklearn.metrics
predictions_dev3=classifier6.predict(feature_matrix_dev3)
print(predictions_dev3)
print(sklearn.metrics.confusion_matrix(dev_labels,predictions_dev3))
print(sklearn.metrics.accuracy_score(dev_labels,predictions_dev3))
```

```
['neg' 'neg' 'pos' ... 'pos' 'pos' 'pos']
[[1945  532]
 [ 383 2140]]
0.817
```

N-gram = 4, C = 0.0005

In [36]:

```
from sklearn.model_selection import train_test_split #split data

train_texts, dev_texts, train_labels, dev_labels=train_test_split(texts,label
vectorizer4=CountVectorizer(max_features=None,binary=True,ngram_range=(4,4))
feature_matrix_train4=vectorizer4.fit_transform(train_texts)
feature_matrix_dev4=vectorizer4.transform(dev_texts)
```

In [37]:

```
print(feature_matrix_train4.shape)
print(feature_matrix_dev4.shape)
```

```
(20000, 3948526)
(5000, 3948526)
```

In [38]:

```
import sklearn.svm #train via classifier
classifier7=sklearn.svm.LinearSVC(C=0.0005,verbose=1)
classifier7.fit(feature_matrix_train4, train_labels)
```

```
[LibLinear]
```

Out[38]:

```
LinearSVC(C=0.0005, verbose=1)
```

In [39]:

```
#Test data
print("DEV",classifier7.score(feature_matrix_dev4, dev_labels))
print("TRAIN",classifier7.score(feature_matrix_train4, train_labels))
```

```
DEV 0.7442
TRAIN 0.9943
```

In [40]:

```
# Test C value and see the results
import sklearn.metrics
predictions_dev4=classifier7.predict(feature_matrix_dev4)
print(predictions_dev4)
print(sklearn.metrics.confusion_matrix(dev_labels,predictions_dev4))
print(sklearn.metrics.accuracy_score(dev_labels,predictions_dev4))
```

```
['neg' 'neg' 'neg' ... 'pos' 'neg' 'pos']
[[2014  452]
 [ 827 1707]]
0.7442
```

N-gram = 4, C = 0.005

In [41]:

```
import sklearn.svm #train via classifier
classifier7=sklearn.svm.LinearSVC(C=0.005,verbose=1)
classifier7.fit(feature_matrix_train4, train_labels)
```

```
[LibLinear]
```

Out[41]:

```
LinearSVC(C=0.005, verbose=1)
```

In [42]:

```
#Test data
print("DEV",classifier7.score(feature_matrix_dev4, dev_labels))
print("TRAIN",classifier7.score(feature_matrix_train4, train_labels))
```

```
DEV 0.767
TRAIN 0.99985
```

In [43]:

```
# Test C value and see the results
import sklearn.metrics
predictions_dev4=classifier7.predict(feature_matrix_dev4)
print(predictions_dev4)
print(sklearn.metrics.confusion_matrix(dev_labels,predictions_dev4))
print(sklearn.metrics.accuracy_score(dev_labels,predictions_dev4))
```

```
['neg' 'neg' 'neg' ... 'pos' 'pos' 'pos']
[[1796  670]
 [ 495 2039]]
0.767
```

Conclusion:

- Lower n-grams will return better accuracy results
- Lower C-value will return better accuracy results

In [ ]:

```

```