Sentiment analysis is a field dedicated to extracting subjective emotions and feelings from text. One commo
expresses negative or positive feelings. Written reviews are great datasets for doing sentiment analysis beca
train an algorithm.

You'll work with the IMDB dataset: a set of 50,000 highly polarized reviews from the Internet Movie Database.
25,000 reviews for testing, each set consisting of 50% negative and 50% positive reviews.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.datasets import imdb
```

The argument num_words=7500 means you'll only keep the top 7500 most fre- quently occurring words in th
allows you to work with vector data of manageable size. The variables train_data and test_data are lists of re
sequence of words)....list of lists! train_labels and test_labels are lists of 0s and 1s, where 0 stands for negati

We will try to convert list of lists into a dataframe. '/content/gdrive/My Drive/Colab Notebooks/IMDB/train/p
contain a sequence (list) of words.

```
vocabulary=7500
```

```
# save np.load
#np_load_old = np.load

# modify the default parameters of np.load
#np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

# call load_data with allow_pickle implicitly set to true
#(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=vocab

# restore np.load for future normal usage
#np.load = np_load_old
```

```
np.load.__defaults__=(None, True, True, 'ASCII')
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=vocabu
np.load.__defaults__=(None, False, True, 'ASCII')
```

Loads the data as a list of integers

```
print(train_data[0])
```

It shows that Tokenizing has been done allready

It shows that Tokenizing has been done allready

```
print(len(train_data[100]))
```

Reviews have already been converted in a sequence of indexed words. One-hot encode the lists to turn them
instance, turning the sequence [3, 5] into a 10,000-dimensional vector that would be all 0s except for indices 3

We define a function called vectorize_sequences that takes 2 arguments: review and dimension of vocabular

```
def vectorize_sequences(sequences, dimension=10000):
  results=np.zeros((len(sequences), dimension)) #matrix of 10000 columns and with rows
  for i, sequence in enumerate(sequences):
    results[i, sequence]=1
  return results
```

```
X_train=vectorize_sequences(train_data)
X_test=vectorize_sequences(test_data)
```

```
print(X_train[0])
print(X_test[0])
```

```
Using TensorFlow backend.
```

```
X_train.shape
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 3
```

```
X_test.shape
```

```
158
```

We currently have a list of lists and will convert to a panda dataframe using the pandas DataFrame construct

```
X_train_df=pd.DataFrame(X_train)
y_train_df=pd.DataFrame(train_labels, columns=['IMDB training labels'])

X_test_df=pd.DataFrame(X_test)
y_test_df=pd.DataFrame(test_labels, columns=['IMDB Testing labels'])
```

```
X_train_df.head()
```

```
[0. 1. 1. ... 0. 0. 0.]
```

```
[0. 1. 1. ... 0. 0. 0.]
```

```
y_train_df['IMDB training labels'].value_counts()
```

```
(25000, 10000)
```

Dataset is balanced

we want to merge the training data and labels to make up 1 dataframe

```
train_df = pd.concat([X_train_df, y_train_df], axis=1) #combined training data and labe
```

```
test_df = pd.concat([X_test_df, y_test_df], axis=1) #combined training data and labels
```

```
train_df.head()
```

```
(25000, 10000)
```

We will use a logistic regression classifier - categorizes data in 2 classes via Sigmoid

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train_df, y_train_df)
score = classifier.score(X_test_df, y_test_df)
print("Accuracy:", score)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0
1	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0
2	0.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0
3	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Logistics Regression gives us an accuracy of 86%

```
from sklearn.naive_bayes import MultinomialNB
```

```
model=MultinomialNB()
```

```
model.fit(X_train_df, y_train_df);
score=model.score(X_test_df, y_test_df)
print("Accuracy:", score)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0
1	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0
2	0.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Naive Bayes - Multinomial gives us an accuracy of 84%

```
from sklearn.naive_bayes import GaussianNB
```

```
model=GaussianNB()
```

```
model.fit(X_train_df, y_train_df);
score=model.score(X_test_df, y_test_df)
print("Accuracy:", score)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432:
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:724: Data
  y = column_or_1d(y, warn=True)
Accuracy: 0.85852
```

Naive Bayes - Gaussian gives us an accuracy of 69% - not a surprise as multinomial assumption is known to l
Gaussian assumption

Now we will look at Random Forests and Adaboost

```
from sklearn.ensemble import RandomForestClassifier
```

```
clf=RandomForestClassifier(n_estimators=100) #n_estimators is the number of trees in t
```

```
clf.fit(X_train_df,y_train_df)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:724: Data
  y = column_or_1d(y, warn=True)
Accuracy: 0.8424
```

```
y_pred_df=clf.predict(X_test_df)
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test_df, y_pred_df))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:724: Data
  y = column_or_1d(y, warn=True)
```

```
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(n_estimators=50,learning_rate=1,random_state=0)
model = clf.fit(X_train_df, y_train_df)
```

```
print("Accuracy:",metrics.accuracy_score(y_test_df, y_pred_df))
```