The IMDB dataset actually comes packaged with keras and its allready tokenized, menaing the text is allready
IMDB dataset contains 50,000 movie reviews (25,000 for training and 25,000 for testing). Each set contains o

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

```
vocabulary=7500 # we will only use the 7500 most frequently used words
```

Next code block is to fix a bug in keras. If bug would not exist this block would be just 1 line of code: (train_da
imdb.load_data(num_words=vocabulary)

I suggest to just copy the fix and it does not really matter if you understand it or not

```
# 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=vocabu

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

In the next line of code we will print the lists that contain sequences of words represented by a word index. If
indices we would need to add one pre-processing step using Tokenizer

```
print(train_data[1]) # train_data is a list of word sequences
```

Now we will vectorize the training and test data. Basically we will create a matrix where the rows are the revie
(7500 columns). We will set a 1 in the correct column if the word of the review matches a word of the vocabu

```
def vectorize_sequences(sequences, dimension=vocabulary):
    results=np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence]=1
    return results
```

Now we apply the function to our training and test data as well as the labels. For the labels we use a different the list to an array and we assign the items in the array to float32

```
x_train=vectorize_sequences(train_data)
x_test=vectorize_sequences(test_data)

y_train=np.asarray(train_labels).astype('float32')
y_test=np.asarray(test_labels).astype('float32')
```

```
pip install bayesian-optimization
```



```
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
import bayes_opt
from bayes_opt import BayesianOptimization
from sklearn.model_selection import cross_val_score
```

```
#model_not_tuned = XGBClassifier()
#model_not_tuned.fit(x_train, y_train)
```

Let us look at score before parameter tuning and using defaults

```
pbounds = {'n_estimators': (50, 1000), 'eta': (0.01, 3), 'max_depth': (1,32), 'gamma':
```

```
model_tuning = XGBClassifier(n_jobs=-1)
```

```
def xgboostcv(eta, n_estimators, max_depth, gamma, min_child_weight, subsample, colsam
    return np.mean(cross_val_score(model_tuning, x_train, y_train, cv=5, scoring='accu
```

```
optimizer = BayesianOptimization(
    f=xgboostcv,
    pbounds=pbounds,
    random_state=1)
```

```
optimizer.maximize(
    init_points=2,
    n_iter=4)
print(optimizer.max)
```

```
model = XGBClassifier(eta=2, n_estimators=138, max_depth=10, min_child_weight=5, gamma
```

```
model.fit(x_train, y_train)
```

```
y_pred = model.predict(x_test)
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
from sklearn.metrics import confusion_matrix
y_pred=model.predict(x_test)
confusion_matrix(y_test,y_pred)
```