The IMDB dataset actually comes packaged with keras and its allready tokenized, menaing the text is allready converted in a sequence of unique word indices. The IMDB dataset contains 50,000 movie reviews (25,000 fo training and 25,000 for testing). Each set contains of 50% positive and 50% negative reviews (12,500 x 2).

```
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 train_labels), (test_data, test_labels) = 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 text has not been converted to a sequence of 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 and where the columns represent the vocabulary (7500 columns). We will set a 1 in the correct column if the of the review matches a word of the vocabulary. This means that matrix will be rather sparse.

```
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 method. We simply use the asarray function to convert the list to an array and we assign the items in the arra 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')
```

Now we are ready to apply Naive Bayes. For this example (discreet) we will use the Multinomial NB algorithm

```
from sklearn.naive_bayes import MultinomialNB # smoothing is automatically applied
model=MultinomialNB()
```

```
model.fit(x_train, train_labels)
score=model.score(x_test, test_labels)
print("Accuracy:", score)
```

Let us take a look at other KPIs by copying some of the code of the KNN exercise on the Pima Indians

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

We note that we have 2354 false negatives and 1586 false positives. Recall in our case is TP/(TP+FN) = 81%. Precission is TP/(TP+FP)=86%

```
from sklearn.metrics import roc_curve
y_pred_proba=model.predict_proba(x_test)[:,1]
fpr, tpr, thresholds=roc_curve(y_test, y_pred_proba)
```

```
plt.plot([0,1], [0,1], '')
plt.plot(fpr, tpr, label='')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('NB Multi ROC curve')
plt.show()
```

We actually see a nice result for this classifier and this is reflected in the high AUC score

```
from sklearn.metrics import roc_auc_score #area under the ROC curve
roc_auc_score(y_test, y_pred_proba)
```