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
In lesson 6 I mentioned that we would try KNN on the IMDB dataset but it is not a good idea because of the h
we only get an accuracy of 50%.

```
import numpy as np
from keras import preprocessing
from keras.datasets import imdb
```

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

Next block of code block has been commented out because it does not work anymore

```
# 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')
```

In the next line of code we will print the lists that contain sequences of words represented by a word index.

```
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
7350 places where we will have a zero. 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
```

```
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 develop our neural network

```
from keras import models
from keras import layers
from keras import optimizers
from keras import losses
from keras import metrics

model=models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(vocabulary,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
#model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

```
#model.compile(optimizer=optimizers.RMSprop(lr=0.001), loss='binary_crossentropy', met
```

```
#model.compile(optimizer=optimizers.RMSprop(lr=0.001), loss=losses.binary_crossentropy
```

No we will set aside a validation set

```
x_val=x_train[:10000]
partial_x_train=x_train[10000:]

y_val=y_train[:10000]
partial_y_train=y_train[10000:]
```

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
```

```
history=model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512, validat
```

```
import matplotlib.pyplot as plt
history_dict=history.history
loss_values=history_dict['loss']
val_loss_values=history_dict['val_loss']
epochs=range(1, 21)
```

```
plt.plot(epochs, loss_values, 'bo', label='Training Loss') #bo is for blue dotted line
plt.plot(epochs, val_loss_values, 'b', label='Training Loss') #b is for blue line
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show
```

```
model=models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(vocabulary,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results=model.evaluate(x_test, y_test)
```