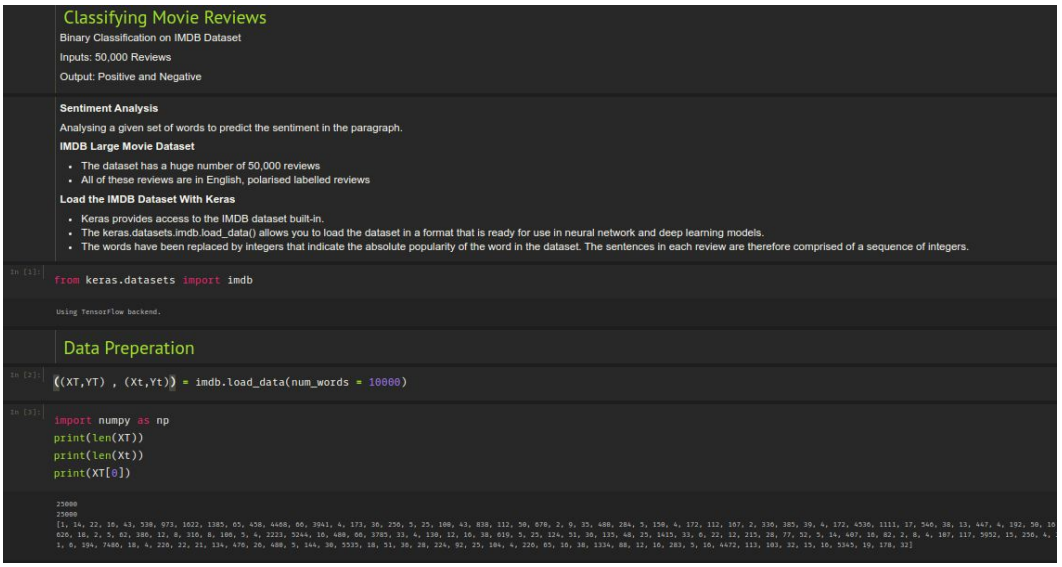
Tushar Goyal, IIT MANDI, CSE

Sentiment Analysis using Neural Networks

So here we used IMDB dataset, and are gonna do binary classification on movie reviews. We are going to import the dataset from Keras, we can download directly from the internet as well.

Data Preparation :



Half of the reviews go in the training set and half of the reviews go in the test set.

- Every review is represented as a list of numbers, it does this according to a mapping which we say vocab
- Vocab gave by Keras, we can see using **imdb.get_word_index() ,** it's given as word to index mapping, we make a reverse lookup map from this to convert review given as list of numbers to actual text.

Our next step is , **To feed the review into MLP**(neural network) , then we should do vectorization of the review,we can't feed variable length reviews into classifier.

- Our vocab size is 10,000 so we make sure every review is vector of len 10k [0...0110010..10] of 0's and 1'st
- So like lets say review is "Movie was amazing" so Movie ki jo mapping hogi vocab me vector me us position ko 1 krdenge , like say Movie is mapped to 113 then 113 is 1 in vector , similarly for all words

```
# Next Step -  Vectorize the Data
# Vocab size - 10,000 we will make every sentence is represented by vector of len 10k [0010100..1....1..0..1..]

def vectorize_sentences(sentences,dim = 10000):
  outputs = np.zeros((len(sentences),10000))

  for i,idx in enumerate(sentences):

    outputs[i,idx] = 1

  return outputs
```

```
X_train = vectorize_sentences(XT)
X_test = vectorize_sentences(Xt)
print(X_train.shape)
print(X_train[0].shape)
X_train[0]
```

```
(25000, 10000)
(10000,)

array([0., 1., 1., ..., 0., 0., 0.])
```

 # Defining Model Architecture

Define Your Model Architecture

- Use fully connected/Dense Layers with ReLu Activation
- 2 Hidden Layers with 16 Units each
- 1 Output layer with 1 unit(Sigmoid function)

```
from keras import models
from keras.layers import Dense
```

```
#define the model
model = models.Sequential()
model.add( Dense ( 16,activation = 'relu',input_shape = (10000,) ) )
model.add(Dense(16,activation = 'relu'))
model.add(Dense(1,activation = 'sigmoid'))
```

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

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 16)                160016
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 17
=================================================================
Total params: 160,305
Trainable params: 160,305
Non-trainable params: 0
_________________________________________________________________
```

 # TRAINING

Training and Validation

Out of 25000 training points ,lets pick first 5000 for validation and 20k for training

```
x_val = X_train[:5000]
x_train_new = X_train[5000:]

y_val = Y_train[:5000]
y_train_new = Y_train[5000:]
```

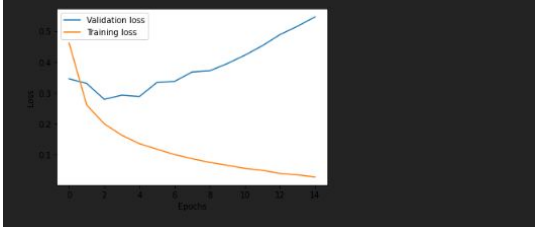
```
hist = model.fit(x_train_new , y_train_new , epochs = 15, batch_size = 512 , validation_data=(x_val,y_val) )
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/15
20000/20000 [==============================] - 2s 81us/step - loss: 0.4603 - accuracy: 0.8052 - val_loss: 0.3445 - val_accuracy: 0.8718
Epoch 2/15
20000/20000 [==============================] - 1s 61us/step - loss: 0.2601 - accuracy: 0.9103 - val_loss: 0.3298 - val_accuracy: 0.8640
Epoch 3/15
20000/20000 [==============================] - 1s 63us/step - loss: 0.1987 - accuracy: 0.9298 - val_loss: 0.2781 - val_accuracy: 0.8886
Epoch 4/15
20000/20000 [==============================] - 1s 64us/step - loss: 0.1618 - accuracy: 0.9452 - val_loss: 0.2916 - val_accuracy: 0.8842
Epoch 5/15
20000/20000 [==============================] - 1s 64us/step - loss: 0.1344 - accuracy: 0.9534 - val_loss: 0.2872 - val_accuracy: 0.8912
Epoch 6/15
20000/20000 [==============================] - 1s 67us/step - loss: 0.1165 - accuracy: 0.9613 - val_loss: 0.3328 - val_accuracy: 0.8806
Epoch 7/15
20000/20000 [==============================] - 1s 66us/step - loss: 0.0993 - accuracy: 0.9671 - val_loss: 0.3362 - val_accuracy: 0.8812
Epoch 8/15
20000/20000 [==============================] - 1s 65us/step - loss: 0.0858 - accuracy: 0.9736 - val_loss: 0.3660 - val_accuracy: 0.8752
Epoch 9/15
20000/20000 [==============================] - 1s 66us/step - loss: 0.0739 - accuracy: 0.9768 - val_loss: 0.3706 - val_accuracy: 0.8836
Epoch 10/15
20000/20000 [==============================] - 1s 65us/step - loss: 0.0644 - accuracy: 0.9804 - val_loss: 0.3938 - val_accuracy: 0.8812
Epoch 11/15
20000/20000 [==============================] - 1s 67us/step - loss: 0.0543 - accuracy: 0.9837 - val_loss: 0.4210 - val_accuracy: 0.8786
Epoch 12/15
20000/20000 [==============================] - 1s 65us/step - loss: 0.0480 - accuracy: 0.9851 - val_loss: 0.4520 - val_accuracy: 0.8774
Epoch 13/15
20000/20000 [==============================] - 1s 64us/step - loss: 0.0381 - accuracy: 0.9902 - val_loss: 0.4879 - val_accuracy: 0.8728
Epoch 14/15
20000/20000 [==============================] - 1s 66us/step - loss: 0.0333 - accuracy: 0.9908 - val_loss: 0.5150 - val_accuracy: 0.8722
Epoch 15/15
20000/20000 [==============================] - 1s 66us/step - loss: 0.0267 - accuracy: 0.9930 - val_loss: 0.5454 - val_accuracy: 0.8736
```

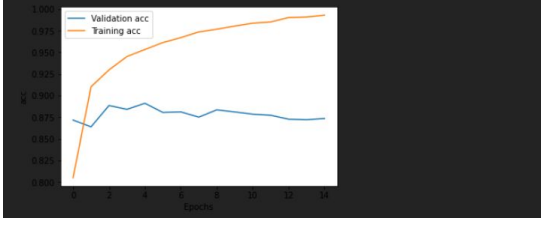
Visualising error and accuracy

```
h = hist.history
```

```
plt.plot(h['val_loss'] , label = "Validation loss")
plt.plot(h['loss'] , label = "Training loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.plot(h['val_accuracy'] , label = "Validation acc")
plt.plot(h['accuracy'] , label = "Training acc")
plt.xlabel("Epochs")
plt.ylabel("acc")
plt.legend()
plt.show()
```



#Accuracy metrics →