In [53]:
```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will
 list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be sa
ved outside of the current session
```

```
/kaggle/input/physics-vs-chemistry-vs-biology/dataset/train.csv
/kaggle/input/physics-vs-chemistry-vs-biology/dataset/test.csv
```

In [54]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models

from sklearn.preprocessing import LabelEncoder
import pickle
import numpy as np
import pandas as pd
np.random.seed(1234)
```

In [55]:
```
train = pd.read_csv('/kaggle/input/physics-vs-chemistry-vs-biology/datase
t/train.csv', header=0, usecols=[1,2], encoding='latin-1')
(trainLen, _ ) = train.shape
for i in range(trainLen):
    if train.loc[i, "Topic"] == "Physics":
        train.loc[i, "Topic"] = 1
    else:
        train.loc[i, "Topic"] = 0

print('rows and columns:', train.shape)
print(train.head())

print("\n\n")



test = pd.read_csv('/kaggle/input/physics-vs-chemistry-vs-biology/datase
t/test.csv', header=0, usecols=[1,2], encoding='latin-1')
(testLen, _ ) = test.shape
for i in range(testLen):
    if test.loc[i, "Topic"] == "Physics":
        test.loc[i, "Topic"] = 1
    else:
        test.loc[i, "Topic"] = 0

print('rows and columns:', test.shape)
print(test.head())
```

```
rows and columns: (8695, 2)
                                             Comment Topic
0  A few things. You might have negative- frequen...     0
1  Is it so hard to believe that there exist part...     1
2                                     There are bees     0
3  I'm a medication technician. And that's alot o...     0
4                     Cesium is such a pretty metal.     0
```

```
rows and columns: (1586, 2)
                                             Comment Topic
0  Personally I have no idea what my IQ is. Iâ  v...     0
1  I'm skeptical. A heavier lid would be needed t...     1
2  I think I have 100 cm of books on the subject....     0
3  Is chemistry hard in uni. Ive read somewhere t...     0
4  In addition to the other comment, you can crit...     1
```

In [56]:

```
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
train data size:  (8695, 2)
test data size:  (1586, 2)
```

In [57]:

```
# set up X and Y
num_labels = 2
vocab_size = 25000
batch_size = 100

# fit the tokenizer on the training data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.Comment)

x_train = tokenizer.texts_to_matrix(train.Comment, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.Comment, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.Topic)
y_train = encoder.transform(train.Topic)
y_test = encoder.transform(test.Topic)

# check shape
print("train shapes:", x_train.shape, y_train.shape)
print("test shapes:", x_test.shape, y_test.shape)
print("test first five labels:", y_test[:5])
```

```
train shapes: (8695, 25000) (8695,)
test shapes: (1586, 25000) (1586,)
test first five labels: [0 1 0 0 1]
```

In [58]:
```
# fit model
model = models.Sequential()
model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='norm
al', activation='relu'))
model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoi
d'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=30,
                    verbose=1,
                    validation_split=0.1)
```

```
Epoch 1/30
79/79 [==============================] - 1s 11ms/step - loss: 0.5593 -
accuracy: 0.7514 - val_loss: 0.4691 - val_accuracy: 0.8011
Epoch 2/30
79/79 [==============================] - 1s 9ms/step - loss: 0.3265 -
accuracy: 0.8657 - val_loss: 0.3784 - val_accuracy: 0.8529
Epoch 3/30
79/79 [==============================] - 1s 9ms/step - loss: 0.2010 -
accuracy: 0.9259 - val_loss: 0.3547 - val_accuracy: 0.8609
Epoch 4/30
79/79 [==============================] - 1s 9ms/step - loss: 0.1351 -
accuracy: 0.9513 - val_loss: 0.3572 - val_accuracy: 0.8609
Epoch 5/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0986 -
accuracy: 0.9683 - val_loss: 0.3718 - val_accuracy: 0.8678
Epoch 6/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0765 -
accuracy: 0.9769 - val_loss: 0.3957 - val_accuracy: 0.8621
Epoch 7/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0625 -
accuracy: 0.9819 - val_loss: 0.4117 - val_accuracy: 0.8575
Epoch 8/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0530 -
accuracy: 0.9831 - val_loss: 0.4337 - val_accuracy: 0.8506
Epoch 9/30
79/79 [==============================] - 1s 8ms/step - loss: 0.0465 -
accuracy: 0.9845 - val_loss: 0.4606 - val_accuracy: 0.8575
Epoch 10/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0415 -
accuracy: 0.9870 - val_loss: 0.4779 - val_accuracy: 0.8540
Epoch 11/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0378 -
accuracy: 0.9881 - val_loss: 0.4963 - val_accuracy: 0.8517
Epoch 12/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0350 -
accuracy: 0.9885 - val_loss: 0.5194 - val_accuracy: 0.8506
Epoch 13/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0325 -
accuracy: 0.9888 - val_loss: 0.5480 - val_accuracy: 0.8483
Epoch 14/30
```

```
79/79 [==============================] - 1s 9ms/step - loss: 0.0306 -
accuracy: 0.9891 - val_loss: 0.5671 - val_accuracy: 0.8471
Epoch 15/30
79/79 [==============================] - 1s 15ms/step - loss: 0.0292 -
accuracy: 0.9894 - val_loss: 0.5864 - val_accuracy: 0.8425
Epoch 16/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0278 -
accuracy: 0.9896 - val_loss: 0.6084 - val_accuracy: 0.8425
Epoch 17/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0266 -
accuracy: 0.9899 - val_loss: 0.6246 - val_accuracy: 0.8437
Epoch 18/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0256 -
accuracy: 0.9904 - val_loss: 0.6503 - val_accuracy: 0.8345
Epoch 19/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0247 -
accuracy: 0.9905 - val_loss: 0.6702 - val_accuracy: 0.8368
Epoch 20/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0239 -
accuracy: 0.9911 - val_loss: 0.6904 - val_accuracy: 0.8333
Epoch 21/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0234 -
accuracy: 0.9912 - val_loss: 0.7149 - val_accuracy: 0.8345
Epoch 22/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0228 -
accuracy: 0.9913 - val_loss: 0.7338 - val_accuracy: 0.8310
Epoch 23/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0222 -
accuracy: 0.9916 - val_loss: 0.7573 - val_accuracy: 0.8299
Epoch 24/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0217 -
accuracy: 0.9918 - val_loss: 0.7728 - val_accuracy: 0.8276
Epoch 25/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0213 -
accuracy: 0.9913 - val_loss: 0.7961 - val_accuracy: 0.8299
Epoch 26/30
79/79 [==============================] - 1s 8ms/step - loss: 0.0210 -
accuracy: 0.9919 - val_loss: 0.8211 - val_accuracy: 0.8276
Epoch 27/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0206 -
accuracy: 0.9922 - val_loss: 0.8263 - val_accuracy: 0.8276
```

```
Epoch 28/30
79/79 [==============================] - 1s 8ms/step - loss: 0.0204 -
accuracy: 0.9919 - val_loss: 0.8601 - val_accuracy: 0.8264
Epoch 29/30
79/79 [==============================] - 1s 8ms/step - loss: 0.0200 -
accuracy: 0.9913 - val_loss: 0.8842 - val_accuracy: 0.8264
Epoch 30/30
79/79 [==============================] - 1s 9ms/step - loss: 0.0196 -
accuracy: 0.9922 - val_loss: 0.8982 - val_accuracy: 0.8276
```

In [59]:

```
# evaluate
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
print(score)
```

```
16/16 [==============================] - 0s 4ms/step - loss: 1.4412 -
accuracy: 0.8670
Accuracy:  0.866960883140564
[1.4412128925323486, 0.866960883140564]
```

In [60]:
```
# get predictions so we can calculate more metrics
pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]
print(pred[:10])
print(pred_labels[:10])
```

```
[[4.5982239e-13]
 [1.7364174e-01]
 [3.2469591e-23]
 [1.1680001e-14]
 [1.3419123e-12]
 [1.5980005e-04]
 [3.6878744e-05]
 [5.9272557e-02]
 [2.5547482e-08]
 [5.8928162e-02]]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

In [61]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score
, f1_score
print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels))
print('recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
accuracy score:  0.8669609079445145
precision score:  0.8328981723237598
recall score:  0.6845493562231759
f1 score:  0.751472320376914
```