Text Classicaition

CS 4395 - Intro to NLP

Dr. Karen Mazidi

Prepare by Leo Nguyen - ldn190002

Instruction 1

```
- Find a text classification data set that interests you.
- Divide into train/test
- Create a graph showing the distribution of the target classes.
- Describe the data set and what the model should be able to predict.
```

Import all require modules an packages

In [1]:
```
# some necessary packages
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import datasets, layers, models, preprocessing

from sklearn.preprocessing import LabelEncoder
import pickle
import numpy as np
import pandas as pd
import seaborn as sb
from fast_ml.model_development import train_valid_test_split

# set seed for reproducibility
np.random.seed(1234)
```

Read the dataset. Only read the: title, text and lable. Ignore the index column in the cvs file

In [2]:
```
df = pd.read_csv('WELFake_Dataset.csv', header=0, usecols=[1,2,3], encoding='latin-1')
print('rows and columns:', df.shape)
print(df.head())
```

```
rows and columns: (72134, 3)
                                               title  \
0  LAW ENFORCEMENT ON HIGH ALERT Following Threat...
1                                                NaN
2  UNBELIEVABLE! OBAMAâ  S ATTORNEY GENERAL SAYS ...
3  Bobby Jindal, raised Hindu, uses story of Chri...
4  SATAN 2: Russia unvelis an image of its terrif...

                                                text  label
0  No comment is expected from Barack Obama Membe...      1
1     Did they post their votes for Hillary already?      1
2   Now, most of the demonstrators gathered last ...      1
3  A dozen politically active pastors came here f...      0
4  The RS-28 Sarmat missile, dubbed Satan 2, will...      1
```

Check the NAs value in entire dataset. Show counts for each column

In [3]:
```
df.isnull().sum()
```

Out[3]:
```
title    558
text      39
label      0
dtype: int64
```

Do some clean up on dataset

Delete rows with NAs. And output the new dimension

In [4]:
```
df = df.dropna()
print('\nDimensions of data frame:', df.shape)
```

```
Dimensions of data frame: (71537, 3)
```

Divide into train/test

In [5]:
```
# split df into train and test
i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
train data size:  (57199, 3)
test data size:  (14338, 3)
```

Graph showing the distribution of the target classes for entire dataset

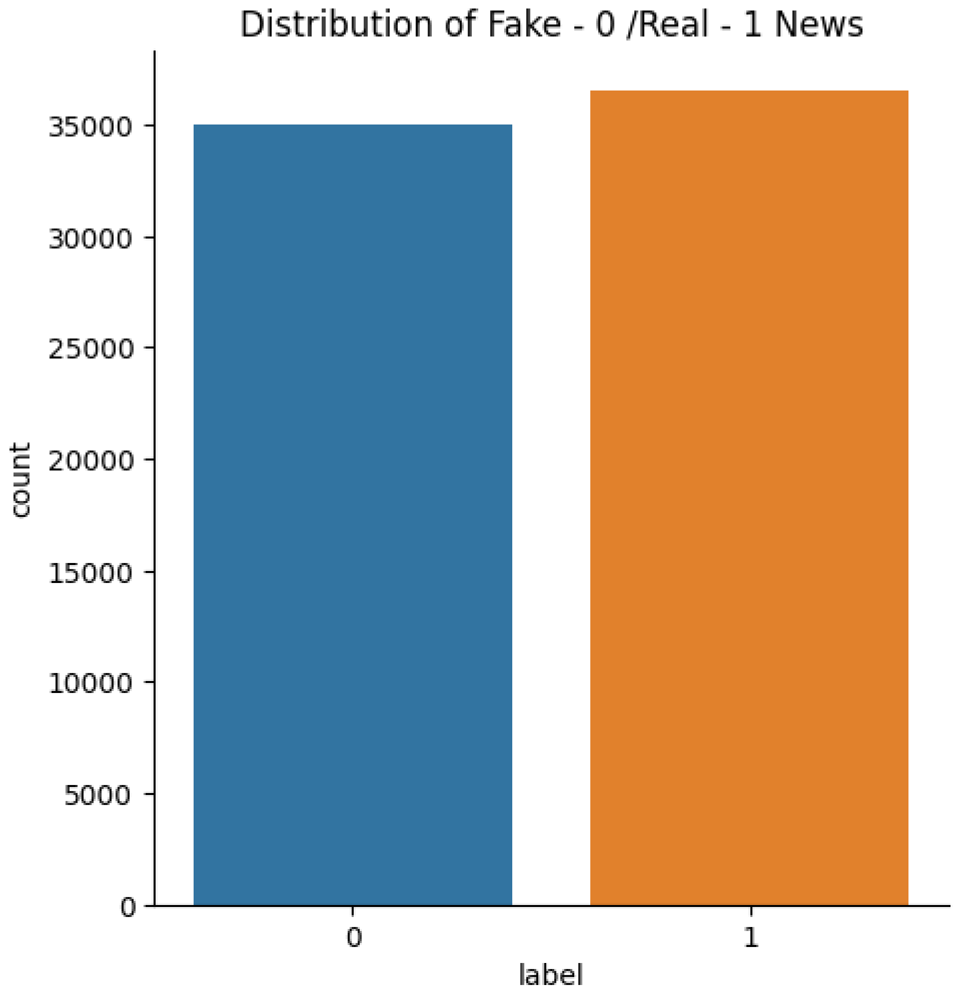
The target class is the **label** column which classify which news is real, which is fake.

Based on the graph, we can see that, we have almost equal amount of fake news and real news on the dataset.

Label (0 = fake and 1 = real).

In [6]:
```
sb.catplot(data=df, x="label", kind='count').set(title='Distribution of Fake - 0 /Real
```

Out[6]: <seaborn.axisgrid.FacetGrid at 0x19d78eb5210>



Describe the data set and what the model should be able to predict.

```
- Fake News Classification:
https://www.kaggle.com/datasets/saurabhshahane/fake-news-
classification

- The dataset contains of 72,134 news articles with 35,028 real and
37,106 fake news. The dataset was merged from four popular news
datasets: Kaggle, McIntire, Reuters, BuzzFeed Political to prevent
over-fitting of classifiers and to provide more text data for better
ML training.

- Dataset contains four columns: Serial number (starting from 0);
Title (about the text news heading); Text (about the news content);
and Label (0 = fake and 1 = real).

- However there is some NAs value inside the original dataset. After
cleaning (Remove NAs value). The number of articles was reduce from
72,134 to 71,537.

- Cleaned dataset is divided into train/test (80/20). Then use the
```

```
train set to build the model to predict which articles is real, which
is fake. (0 = fake and 1 = real).

 - The models only use "text" column as a predictor to predict
"label" columns.
```

Intruction 2 - Sequential Model

```
- Create a sequential model
- Evaluate on the test data
```

In [7]:
```
df.head()
```

Out[7]:

	title	text	label
0	LAW ENFORCEMENT ON HIGH ALERT Following Threat...	No comment is expected from Barack Obama Membe...	1
2	UNBELIEVABLE! OBAMAâ S ATTORNEY GENERAL SAYS ...	Now, most of the demonstrators gathered last ...	1
3	Bobby Jindal, raised Hindu, uses story of Chri...	A dozen politically active pastors came here f...	0
4	SATAN 2: Russia unvelis an image of its terrif...	The RS-28 Sarmat missile, dubbed Satan 2, will...	1
5	About Time! Christian Group Sues Amazon and SP...	All we can say on this one is it s about time ...	1

Create a sequential model

Prepare the data before building model

In [8]:
```
# 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.text) # Only use text column as the predictor for the moc

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

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

# 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: (57199, 25000) (57199,)
test shapes: (14338, 25000) (14338,)
test first five labels: [1 1 1 1 1]
```

Build the model

In [9]:

```
# fit model
model1 = models.Sequential()
model1.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='normal', activat
model1.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))

# compile
model1.compile(loss='binary_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])

# train
history = model1.fit(x_train, y_train,
                     batch_size=batch_size,
                     epochs=30,
                     verbose=1,
                     validation_split=0.1)
```

```
Epoch 1/30
515/515 [==============================] - 35s 63ms/step - loss: 0.1646 - accuracy:
0.9462 - val_loss: 0.0894 - val_accuracy: 0.9722
Epoch 2/30
515/515 [==============================] - 16s 31ms/step - loss: 0.0296 - accuracy:
0.9930 - val_loss: 0.0889 - val_accuracy: 0.9771
Epoch 3/30
515/515 [==============================] - 8s 15ms/step - loss: 0.0101 - accuracy: 0.
9988 - val_loss: 0.0948 - val_accuracy: 0.9759
Epoch 4/30
515/515 [==============================] - 8s 15ms/step - loss: 0.0073 - accuracy: 0.
9991 - val_loss: 0.1256 - val_accuracy: 0.9753
Epoch 5/30
515/515 [==============================] - 8s 15ms/step - loss: 0.0066 - accuracy: 0.
9990 - val_loss: 0.1296 - val_accuracy: 0.9752
Epoch 6/30
515/515 [==============================] - 8s 15ms/step - loss: 0.0074 - accuracy: 0.
9986 - val_loss: 0.1387 - val_accuracy: 0.9724
Epoch 7/30
515/515 [==============================] - 8s 15ms/step - loss: 0.0065 - accuracy: 0.
9988 - val_loss: 0.1529 - val_accuracy: 0.9743
Epoch 8/30
515/515 [==============================] - 8s 15ms/step - loss: 0.0095 - accuracy: 0.
9985 - val_loss: 0.1529 - val_accuracy: 0.9740
Epoch 9/30
515/515 [==============================] - 8s 15ms/step - loss: 0.0059 - accuracy: 0.
9990 - val_loss: 0.1542 - val_accuracy: 0.9745
Epoch 10/30
515/515 [==============================] - 8s 15ms/step - loss: 0.0029 - accuracy: 0.
9996 - val_loss: 0.1637 - val_accuracy: 0.9745
Epoch 11/30
515/515 [==============================] - 8s 15ms/step - loss: 0.0031 - accuracy: 0.
9995 - val_loss: 0.1967 - val_accuracy: 0.9715
Epoch 12/30
515/515 [==============================] - 8s 15ms/step - loss: 0.0040 - accuracy: 0.
9993 - val_loss: 0.1745 - val_accuracy: 0.9743
Epoch 13/30
515/515 [==============================] - 8s 15ms/step - loss: 0.0014 - accuracy: 0.
9998 - val_loss: 0.1796 - val_accuracy: 0.9750
Epoch 14/30
515/515 [==============================] - 8s 15ms/step - loss: 9.5645e-04 - accurac
y: 0.9999 - val_loss: 0.2068 - val_accuracy: 0.9726
Epoch 15/30
515/515 [==============================] - 8s 15ms/step - loss: 3.8724e-04 - accurac
y: 1.0000 - val_loss: 0.2020 - val_accuracy: 0.9757
Epoch 16/30
515/515 [==============================] - 8s 15ms/step - loss: 3.1645e-04 - accurac
y: 1.0000 - val_loss: 0.2115 - val_accuracy: 0.9752
Epoch 17/30
515/515 [==============================] - 8s 15ms/step - loss: 2.7281e-04 - accurac
y: 1.0000 - val_loss: 0.2213 - val_accuracy: 0.9750
Epoch 18/30
515/515 [==============================] - 8s 15ms/step - loss: 2.3997e-04 - accurac
y: 1.0000 - val_loss: 0.2313 - val_accuracy: 0.9752
Epoch 19/30
515/515 [==============================] - 8s 15ms/step - loss: 2.1521e-04 - accurac
y: 1.0000 - val_loss: 0.2399 - val_accuracy: 0.9750
```

```
Epoch 20/30
515/515 [==============================] - 8s 15ms/step - loss: 1.9436e-04 - accurac
y: 1.0000 - val_loss: 0.2496 - val_accuracy: 0.9748
Epoch 21/30
515/515 [==============================] - 8s 15ms/step - loss: 1.7862e-04 - accurac
y: 1.0000 - val_loss: 0.2589 - val_accuracy: 0.9748
Epoch 22/30
515/515 [==============================] - 8s 16ms/step - loss: 1.6451e-04 - accurac
y: 1.0000 - val_loss: 0.2679 - val_accuracy: 0.9747
Epoch 23/30
515/515 [==============================] - 8s 15ms/step - loss: 1.5245e-04 - accurac
y: 1.0000 - val_loss: 0.2791 - val_accuracy: 0.9747
Epoch 24/30
515/515 [==============================] - 8s 15ms/step - loss: 1.4418e-04 - accurac
y: 1.0000 - val_loss: 0.2900 - val_accuracy: 0.9748
Epoch 25/30
515/515 [==============================] - 8s 15ms/step - loss: 1.3509e-04 - accurac
y: 1.0000 - val_loss: 0.2983 - val_accuracy: 0.9743
Epoch 26/30
515/515 [==============================] - 8s 15ms/step - loss: 1.2891e-04 - accurac
y: 1.0000 - val_loss: 0.3089 - val_accuracy: 0.9745
Epoch 27/30
515/515 [==============================] - 8s 15ms/step - loss: 1.2458e-04 - accurac
y: 1.0000 - val_loss: 0.3169 - val_accuracy: 0.9743
Epoch 28/30
515/515 [==============================] - 8s 15ms/step - loss: 1.1880e-04 - accurac
y: 1.0000 - val_loss: 0.3292 - val_accuracy: 0.9750
Epoch 29/30
515/515 [==============================] - 8s 15ms/step - loss: 1.1579e-04 - accurac
y: 1.0000 - val_loss: 0.3344 - val_accuracy: 0.9740
Epoch 30/30
515/515 [==============================] - 8s 15ms/step - loss: 1.1046e-04 - accurac
y: 1.0000 - val_loss: 0.3474 - val_accuracy: 0.9747
```

In [10]:
```python
# Show model summary
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                800032

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 800,065
Trainable params: 800,065
Non-trainable params: 0
_________________________________________________________________
```

Evaluation

Use model to predict the target on test data

In [11]:
```python
# get predictions so we can calculate more metrics
pred = model1.predict(x_test)
```

```
pred_labels = [1 if p>0.5 else 0 for p in pred]
pred[:10]
```

```
449/449 [==============================] - 3s 4ms/step
```

Out[11]:
```
array([[1.0000000e+00],
       [1.0000000e+00],
       [1.0000000e+00],
       [1.0000000e+00],
       [9.9999946e-01],
       [4.2885760e-24],
       [1.0620265e-26],
       [1.0000000e+00],
       [1.0000000e+00],
       [1.0000000e+00]], dtype=float32)
```

Evaluate and calculate the metrics

In [12]:
```
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:  0.9745431719905148
precision score:  0.9727137646899905
recall score:  0.9778652906029331
f1 score:  0.9752827249949211
```

Instruction 3 - Sequential Model with RNN, CNN Architecture

Try different architecture like RNN, CNN, etc and evaluate on the test data

RNN Architecture

Prepare the data before building model

In [13]:
```
# Try different set up for X and Y
max_features = 10000
maxlen = 500
batch_size = 32


# pad the data to maxlen
train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

Build model with RNN Architecture

In [14]:
```
# build a Sequential model with Embedding and SimpleRNN layers

model2 = models.Sequential()
model2.add(layers.Embedding(max_features, 32))
model2.add(layers.SimpleRNN(32))
model2.add(layers.Dense(1, activation='sigmoid'))
```

```
# compile
model2.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# train
history = model2.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
358/358 [==============================] - 46s 124ms/step - loss: 0.6938 - accuracy:
0.5047 - val_loss: 0.6933 - val_accuracy: 0.4938
Epoch 2/10
358/358 [==============================] - 43s 119ms/step - loss: 0.6934 - accuracy:
0.5079 - val_loss: 0.6925 - val_accuracy: 0.5122
Epoch 3/10
358/358 [==============================] - 43s 119ms/step - loss: 0.6936 - accuracy:
0.5063 - val_loss: 0.6923 - val_accuracy: 0.5095
Epoch 4/10
358/358 [==============================] - 43s 119ms/step - loss: 0.6937 - accuracy:
0.5063 - val_loss: 0.6926 - val_accuracy: 0.5109
Epoch 5/10
358/358 [==============================] - 44s 122ms/step - loss: 0.6929 - accuracy:
0.5094 - val_loss: 0.6940 - val_accuracy: 0.4932
Epoch 6/10
358/358 [==============================] - 43s 120ms/step - loss: 0.6932 - accuracy:
0.5030 - val_loss: 0.6928 - val_accuracy: 0.5106
Epoch 7/10
358/358 [==============================] - 42s 118ms/step - loss: 0.6930 - accuracy:
0.5102 - val_loss: 0.6961 - val_accuracy: 0.5096
Epoch 8/10
358/358 [==============================] - 42s 116ms/step - loss: 0.6935 - accuracy:
0.5049 - val_loss: 0.6942 - val_accuracy: 0.5097
Epoch 9/10
358/358 [==============================] - 42s 116ms/step - loss: 0.6931 - accuracy:
0.5020 - val_loss: 0.6931 - val_accuracy: 0.5099
Epoch 10/10
358/358 [==============================] - 41s 115ms/step - loss: 0.6936 - accuracy:
0.5059 - val_loss: 0.6921 - val_accuracy: 0.5097
```

In [15]:

```
# Show model summary
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 32)          320000

 simple_rnn (SimpleRNN)      (None, 32)                2080

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
_________________________________________________________________
```

Evaluation of RNN

Evaluate and calculate the metrics

In [16]:
```python
from sklearn.metrics import classification_report

# get predictions so we can calculate more metrics
pred = model2.predict(test_data)
pred_labels = [1 if p>0.5 else 0 for p in pred]
print(classification_report(y_test, pred_labels))
```

```
449/449 [==============================] - 10s 22ms/step
              precision    recall  f1-score   support

           0       0.52      0.02      0.03      6974
           1       0.51      0.99      0.68      7364

    accuracy                           0.51     14338
   macro avg       0.52      0.50      0.35     14338
weighted avg       0.52      0.51      0.36     14338
```

CNN Architecture

Build model with CNN Architecture

In [17]:
```python
# build a Sequential model 1D convnet

model3 = models.Sequential()
model3.add(layers.Embedding(max_features, 128, input_length=maxlen))
model3.add(layers.Conv1D(32, 7, activation='relu'))
model3.add(layers.MaxPooling1D(5))
model3.add(layers.Conv1D(32, 7, activation='relu'))
model3.add(layers.GlobalMaxPooling1D())
model3.add(layers.Dense(1))

# compile
model3.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),  # set learn
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```python
# train
history = model3.fit(train_data,
                     y_train,
                     epochs=10,
                     batch_size=128,
                     validation_split=0.2)
```

```
Epoch 1/10
358/358 [==============================] - 116s 323ms/step - loss: 0.7252 - accuracy:
0.4977 - val_loss: 0.6930 - val_accuracy: 0.4983
Epoch 2/10
358/358 [==============================] - 115s 320ms/step - loss: 0.6935 - accuracy:
0.5026 - val_loss: 0.6928 - val_accuracy: 0.5087
Epoch 3/10
358/358 [==============================] - 114s 317ms/step - loss: 0.6936 - accuracy:
0.5006 - val_loss: 0.6928 - val_accuracy: 0.5087
Epoch 4/10
358/358 [==============================] - 113s 317ms/step - loss: 0.6936 - accuracy:
0.5025 - val_loss: 0.6934 - val_accuracy: 0.4963
Epoch 5/10
358/358 [==============================] - 114s 317ms/step - loss: 0.6934 - accuracy:
0.5018 - val_loss: 0.6932 - val_accuracy: 0.5087
Epoch 6/10
358/358 [==============================] - 116s 324ms/step - loss: 0.6933 - accuracy:
0.5002 - val_loss: 0.6926 - val_accuracy: 0.5087
Epoch 7/10
358/358 [==============================] - 116s 324ms/step - loss: 0.6932 - accuracy:
0.5040 - val_loss: 0.6948 - val_accuracy: 0.4962
Epoch 8/10
358/358 [==============================] - 114s 318ms/step - loss: 0.6929 - accuracy:
0.5065 - val_loss: 0.6945 - val_accuracy: 0.4959
Epoch 9/10
358/358 [==============================] - 114s 318ms/step - loss: 0.6930 - accuracy:
0.5044 - val_loss: 0.6937 - val_accuracy: 0.5087
Epoch 10/10
358/358 [==============================] - 113s 317ms/step - loss: 0.6931 - accuracy:
0.5047 - val_loss: 0.6924 - val_accuracy: 0.4960
```

In [18]:

```python
# Show model summary
model3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_1 (Embedding)     (None, 500, 128)          1280000   
                                                                 
 conv1d (Conv1D)             (None, 494, 32)           28704     
                                                                 
 max_pooling1d (MaxPooling1D  (None, 98, 32)           0         
 )                                                               
                                                                 
 conv1d_1 (Conv1D)           (None, 92, 32)            7200      
                                                                 
 global_max_pooling1d (Globa  (None, 32)               0         
 lMaxPooling1D)                                                  
                                                                 
 dense_3 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0
_________________________________________________________________
```

Evaluation of CNN

Evaluate and calculate the metrics

In [19]:
```python
from sklearn.metrics import classification_report

# get predictions so we can calculate more metrics
pred = model3.predict(test_data)
pred_labels = [1 if p>0.5 else 0 for p in pred]
print(classification_report(y_test, pred_labels))
```

```
449/449 [==============================] - 7s 16ms/step
              precision    recall  f1-score   support

           0       0.49      0.97      0.65      6974
           1       0.54      0.04      0.07      7364

    accuracy                           0.49     14338
   macro avg       0.51      0.50      0.36     14338
weighted avg       0.51      0.49      0.35     14338
```

Instruction 4 - Model with Embedded Layer Approaches

```
- Try different embedding approaches and evaluate on the test data
```

Preprocessing data before building and feeding into model

In [20]:
```python
df.head()
```

Out[20]:

	title	text	label
0	LAW ENFORCEMENT ON HIGH ALERT Following Threat...	No comment is expected from Barack Obama Membe...	1
2	UNBELIEVABLE! OBAMAâ S ATTORNEY GENERAL SAYS ...	Now, most of the demonstrators gathered last ...	1
3	Bobby Jindal, raised Hindu, uses story of Chri...	A dozen politically active pastors came here f...	0
4	SATAN 2: Russia unvelis an image of its terrif...	The RS-28 Sarmat missile, dubbed Satan 2, will...	1
5	About Time! Christian Group Sues Amazon and SP...	All we can say on this one is it s about time ...	1

In [21]:

```
# Removing the title from dataset
df1 = df.drop(columns=['title'])
df1.head()
```

Out[21]:

	text	label
0	No comment is expected from Barack Obama Membe...	1
2	Now, most of the demonstrators gathered last ...	1
3	A dozen politically active pastors came here f...	0
4	The RS-28 Sarmat missile, dubbed Satan 2, will...	1
5	All we can say on this one is it s about time ...	1

The original size of dataset is too big for my computer to process.

Try with google colab. It is still not be able to process unless i upgrage to Pro Version

==> Reduce the dataset to 1/3 of its original. So it could be process with my own computer

In [22]:

```
df1 = df1.sample(frac=0.3)  # Reduce dataset to 1/3 of it original size
```

Encoding and decoding the string data so it could be vectorizer later

In [23]:

```
df1['text'] = df1['text'].apply(lambda x: x.encode("latin-1").decode("ascii","ignore")
```

Split into train/validation/test using fast_ml package

In [24]:

```
train_samples, train_labels, val_samples, val_labels, test_samples, test_labels = trai
                                                                               train_size

print('Labels sizes of train, validation, test:', len(train_labels), len(val_labels),
print('Samples sizes of train, validation, test:', len(train_samples), len(val_samples
```

```
Labels sizes of train, validation, test: 17168 2146 2147
Samples sizes of train, validation, test: 17168 2146 2147
```

In [25]:

```
train_samples.shape
```

Out[25]: (17168, 1)

In [26]:
```
train_labels.shape
```

Out[26]: (17168,)

Set up the vectorizer

```
- Use Keras's TextVectorization() function to vectorize the data,
using only the top 20K words. Each sample will be truncated or padded
to a length of 200
```

In [27]:
```
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization

vectorizer = TextVectorization(max_tokens=20000, output_sequence_length=200)
text_ds = tf.data.Dataset.from_tensor_slices(train_samples).batch(128)
vectorizer.adapt(text_ds)
```

In [28]:
```
# create a word index dictionary in which words map to indices
voc = vectorizer.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))
len(word_index)
```

Out[28]: 20000

Building the model:

```
- The model will included:

            - Several layers of Conv1D followed by pooling.
            - Ending in a softmax classification layer.
            - Instead of the usual Keras syntax, this example uses
syntax from the Functional API:
```

Set up the embedding layer

In [29]:
```
EMBEDDING_DIM = 128
MAX_SEQUENCE_LENGTH = 200

embedding_layer = layers.Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            input_length=MAX_SEQUENCE_LENGTH)
```

Building model using syntax from the Functional API

In [30]:
```
# add more layers

int_sequences_input = keras.Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Conv1D(128, 5, activation="relu")(embedded_sequences)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(128, 5, activation="relu")(x)
```

```
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(128, 5, activation="relu")(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation="relu")(x)
x = layers.Dropout(0.5)(x)
preds = layers.Dense(1, activation="softmax")(x)
model4 = keras.Model(int_sequences_input, preds)
model4.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, None)]            0         
                                                                 
 embedding_2 (Embedding)     (None, None, 128)         2560128   
                                                                 
 conv1d_2 (Conv1D)           (None, None, 128)         82048     
                                                                 
 max_pooling1d_1 (MaxPooling  (None, None, 128)        0         
 1D)                                                             
                                                                 
 conv1d_3 (Conv1D)           (None, None, 128)         82048     
                                                                 
 max_pooling1d_2 (MaxPooling  (None, None, 128)        0         
 1D)                                                             
                                                                 
 conv1d_4 (Conv1D)           (None, None, 128)         82048     
                                                                 
 global_max_pooling1d_1 (Glo  (None, 128)              0         
 balMaxPooling1D)                                                
                                                                 
 dense_4 (Dense)             (None, 128)               16512     
                                                                 
 dropout (Dropout)           (None, 128)               0         
                                                                 
 dense_5 (Dense)             (None, 1)                 129       
                                                                 
=================================================================
Total params: 2,822,913
Trainable params: 2,822,913
Non-trainable params: 0
_________________________________________________________________
```

Vectorize train and validation sets

Converting the samples/predictor of train and validation to list of string before vectorizer

In [31]:
```
type(val_samples)
```

Out[31]:
```
pandas.core.frame.DataFrame
```

In [32]:
```
val_samples.head()
```

Out[32]:

	text
4702	If a grand jury indicts Donald Trump, one Fox ...
30075	Share on Facebook Share on Twitter This is som...
7079	One way or another, she s going to have to fa...
43755	WASHINGTON (Reuters) - The U.S. Supreme Court ...
11634	Mises.org October 28, 2016 \nA recent op-ed pi...

Converting the train and validation dataframe to list of string so it can be convert to numpy array then vecterizor

In [33]:
```
x_train = train_samples.text.values.tolist()
x_val = val_samples.text.values.tolist()
```

Vectorizer the data to right-pad the samples

In [34]:
```
x_train = vectorizer(np.array([[s] for s in x_train])).numpy()
x_val = vectorizer(np.array([[s] for s in x_val])).numpy()

y_train = np.array(train_labels)
y_val = np.array(val_labels)
```

In [35]:
```
print('Size of train sample after padding:', len(x_train))
print('Size of train label after padding:', len(y_train))
```

```
Size of train sample after padding: 17168
Size of train label after padding: 17168
```

In [36]:
```
print('Size of val sample after padding:', len(x_val))
print('Size of val label after padding:', len(y_val))
```

```
Size of val sample after padding: 2146
Size of val label after padding: 2146
```

Train the embedded layer model

```
- Binary_crossentropy is used because the final layer is have only
either 0 or 1.
```

In [37]:
```
# compile
model4.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["acc"])

# train
model4.fit(x_train, y_train,
          batch_size=128,
          epochs=20,
          validation_data=(x_val, y_val))
```

```
Epoch 1/20
135/135 [==============================] - 33s 232ms/step - loss: 0.3739 - acc: 0.508
6 - val_loss: 0.1804 - val_acc: 0.4902
Epoch 2/20
135/135 [==============================] - 31s 229ms/step - loss: 0.1117 - acc: 0.508
6 - val_loss: 0.1606 - val_acc: 0.4902
Epoch 3/20
135/135 [==============================] - 31s 232ms/step - loss: 0.0561 - acc: 0.508
6 - val_loss: 0.2385 - val_acc: 0.4902
Epoch 4/20
135/135 [==============================] - 32s 239ms/step - loss: 0.0316 - acc: 0.508
6 - val_loss: 0.1725 - val_acc: 0.4902
Epoch 5/20
135/135 [==============================] - 32s 234ms/step - loss: 0.0142 - acc: 0.508
6 - val_loss: 0.2503 - val_acc: 0.4902
Epoch 6/20
135/135 [==============================] - 31s 228ms/step - loss: 0.0115 - acc: 0.508
6 - val_loss: 0.2937 - val_acc: 0.4902
Epoch 7/20
135/135 [==============================] - 31s 228ms/step - loss: 0.0072 - acc: 0.508
6 - val_loss: 0.3531 - val_acc: 0.4902
Epoch 8/20
135/135 [==============================] - 31s 230ms/step - loss: 0.0023 - acc: 0.508
6 - val_loss: 0.4437 - val_acc: 0.4902
Epoch 9/20
135/135 [==============================] - 31s 228ms/step - loss: 0.0078 - acc: 0.508
6 - val_loss: 0.5241 - val_acc: 0.4902
Epoch 10/20
135/135 [==============================] - 31s 228ms/step - loss: 0.0012 - acc: 0.508
6 - val_loss: 0.6783 - val_acc: 0.4902
Epoch 11/20
135/135 [==============================] - 31s 227ms/step - loss: 0.0092 - acc: 0.508
6 - val_loss: 0.6510 - val_acc: 0.4902
Epoch 12/20
135/135 [==============================] - 31s 228ms/step - loss: 0.0022 - acc: 0.508
6 - val_loss: 0.6193 - val_acc: 0.4902
Epoch 13/20
135/135 [==============================] - 31s 228ms/step - loss: 0.0044 - acc: 0.508
6 - val_loss: 0.5316 - val_acc: 0.4902
Epoch 14/20
135/135 [==============================] - 31s 228ms/step - loss: 0.0020 - acc: 0.508
6 - val_loss: 0.6765 - val_acc: 0.4902
Epoch 15/20
135/135 [==============================] - 31s 227ms/step - loss: 9.3415e-04 - acc:
0.5086 - val_loss: 0.7531 - val_acc: 0.4902
Epoch 16/20
135/135 [==============================] - 31s 230ms/step - loss: 0.0017 - acc: 0.508
6 - val_loss: 0.8555 - val_acc: 0.4902
Epoch 17/20
135/135 [==============================] - 31s 233ms/step - loss: 4.4786e-04 - acc:
0.5086 - val_loss: 1.0549 - val_acc: 0.4902
Epoch 18/20
135/135 [==============================] - 31s 227ms/step - loss: 0.0024 - acc: 0.508
6 - val_loss: 1.0349 - val_acc: 0.4902
Epoch 19/20
135/135 [==============================] - 31s 232ms/step - loss: 0.0086 - acc: 0.508
6 - val_loss: 1.0098 - val_acc: 0.4902
```

```
Epoch 20/20
135/135 [==============================] - 31s 232ms/step - loss: 8.4002e-04 - acc: 0.5086 - val_loss: 0.9996 - val_acc: 0.4902
```

Out[37]:
```
<keras.callbacks.History at 0x19d1236ebf0>
```

Evaluation using Embedded_layer approach

Converting test_sample dataframe to list of string

Vectorizer the data to right-pad the samples

In [38]:
```
test_x = test_samples.text.values.tolist()
test_x = vectorizer(np.array([[s] for s in test_x])).numpy()
```

Evaluate and calculate the metrics

In [39]:
```
from sklearn.metrics import classification_report

preds = model4.predict(test_x)
pred_labels = [np.argmax(p) for p in preds]
print(classification_report(test_labels, pred_labels))
```

```
68/68 [==============================] - 1s 17ms/step
              precision    recall  f1-score   support

           0       0.49      1.00      0.65      1045
           1       0.00      0.00      0.00      1102

    accuracy                           0.49      2147
   macro avg       0.24      0.50      0.33      2147
weighted avg       0.24      0.49      0.32      2147
```

Instruction 4

```
- Analysis of the performance of various approaches


- Among the 4 different approaches: Dense Sequential, RNN, CNN and
Embedded Layer. The 1st approach-Dense Sequential have the most
accuracy number (97%). That is a very high result even though we only
use simple one hidden layer with 32 nodes. It is becauase Dense
Sequential performs a matrix-vector multiplication which can learns
features from all combinational features of the previous layer.

- The 2nd approach-RNN have its accuracy drop significantly compare
to Dense Sequential from 97% to 51%. It is because event though RNN
is more suitable for text compate to CNN. It will not perform well on
a long sequence of text which is exactly the case of this dataset
example when we have around 20,000 word vocabulary.

- The 3nd approach-CNN also have a very low accuracy which is 49%.
This number is even lower than the RNN because, CNN usualy work
```

better on images or video processing. And our dataset is purely string of text.

- The last approach-Embedded Layer also have a very low and simlar accuracy to CNN which is 49%. It is because this approach is very hard to train, especially the softmax function was used as activartor. Moreover, the poor performance also come from the fact that have a very large categories - size of vocabulary = word_index = 20,000. Together with this large number and the nature of word after vectorizer is usually not distributed uniformally lead to the insufficient utilizing of vector.