# Fake News Classification with Tensorflow

## Intro:

Following the previous blog post on image classification, we would be using tensorflow again to perform machine learning tasks —— on text data this time.

While Plague Inc. went 'viral' again during the beginning of the COVID-19 outbreak due to public attention on the pandemics, my favourite game scenario in Plague In. has always been the fake news mode. Albeit some headlines are straight-out troll, the existence of such scenario still speaks volume about how the deliberate use of modern technology and psychological tricks could be used to infect the world with false information and cause extreme consequences to the democracy and health of the soceity.

In this blog, we would be getting some hand-on combat experience against fake news through the creation of a ML & N-Gram based fake news classification model.

## Data acquisition

The following data is a small segment of a Kaggle fakenews dataset. Each row of the data corresponds to an article. The title column gives the title of the article, while the text column gives the full article text. The final column, called fake, is 0 if the article is true and 1 if the article contains fake news, as determined by the authors of the paper：

*Ahmed H, Traore I, Saad S. (2017) "Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques. In: Traore I., Woungang I., Awad A. (eds) Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments. ISDDC 2017. Lecture Notes in Computer Science, vol 10618. Springer, Cham (pp. 127-138).*

```
import plotly.io as pio
```

```
pio.renderers.default = "notebook_connected"
```

```python
import pandas as pd
import numpy as np

train_url = "https://github.com/PhilChodrow/PIC16b/blob/master/datasets/fake_news_train.csv?raw=true"
news = pd.read_csv(train_url)
```

```python
import re
import string

from tensorflow.keras import layers
from tensorflow.keras import losses
```

```python
print(news.dtypes)
news.head()
```

```
Unnamed: 0     int64
title         object
text          object
fake           int64
dtype: object
```

| | Unnamed: 0 | title | text | fake |
|---|---|---|---|---|
| **0** | 17366 | Merkel: Strong result for Austria's FPO 'big c... | German Chancellor Angela Merkel said on Monday... | 0 |
| **1** | 5634 | Trump says Pence will lead voter fraud panel | WEST PALM BEACH, Fla.President Donald Trump sa... | 0 |
| **2** | 17487 | JUST IN: SUSPECTED LEAKER and “Close Confidant... | On December 5, 2017, Circa s Sara Carter warne... | 1 |
| **3** | 12217 | Thyssenkrupp has offered help to Argentina ove... | Germany s Thyssenkrupp, has offered assistance... | 0 |
| **4** | 5535 | Trump say appeals court decision on travel ban... | President Donald Trump on Thursday called the ... | 0 |

# Create Dataset

To parse the text and perform topic analysis, the first step is removing stopwords, i.e., uninformative words like 'and','the','a', etc. We make use of the nltk library to perform such task.

```
import tensorflow as tf
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
True
```

```
from nltk.corpus import stopwords
```

Next, we will create a tensorflow dataset to host our news data. Tensorflow datasets are iterable and well-integrated with the machine learning pipeline. We would write a create_database function that does the following 2 things:

1. Remove stopwords from the article text and title.

2. Construct and return a tf.data.Dataset with two inputs and one output. The input should be of the form (title, text), and the output should consist only of the fake column.

For a tf dataset: - Elements refer to a single output from calling next() on a dataset iterator. Elements may be nested structures containing multiple components. For example, the element (1, (3, "apple")) has one tuple nested in another tuple. The components are 1, 3, and "apple".
- Components refers to the leaves in the nested structure of an element. - To set up a dataset for ML training, we need something in this format:
ds = tf.data.Dataset.from_tensor_slices((features_dict, labels))

```
def make_dataset(df):
```

```
  stop = stopwords.words('english')
  # retain the rest of words, separated by space
  df['text'] = df['text'].apply(lambda x: ' '.join([word for word in x.split()
                                                      if word not in (stop)]))
  df['title'] = df['title'].apply(lambda x: ' '.join([word for word in x.split()
                                                      if word not in (stop)]))
  # Construct tf dataset
  tfds = tf.data.Dataset.from_tensor_slices(({"title":df[["title"]],"text":df["text"]},#feature_dict
            {'fake':df[["fake"]]}))#labels

  # batch
  tfds = tfds.batch(100)
  return tfds
```

```
ds = make_dataset(news)
```

```
for idx,lbl in ds.take(1): # similar to data[:5]
  print(idx['text'])
  print(idx['title'])
  print(lbl)
```

## Validation Data

After constructing the primary dataset, we split off 20% for validation using skip and take.

```
val_size = int(0.2 * len(ds))

ds = ds.shuffle(buffer_size = len(ds))
val_ds = ds.take(val_size)
train_ds = ds.skip(val_size)
```

```
len(val_ds),len(train_ds)
```

```
(45, 180)
```

## Base Rate

Base rate refers to the accuracy of a model that always makes the same guess. Determine the base rate for this data set by examining the labels on the training set —— the base rate would be the proportion of the label with the highest frequency in the label pool: - 1: fake - 0: non-fake

In this case, the base rate is **0.5230** (52.30% of the entries are fake news in this dataset.)

```
news.fake.value_counts()/len(news)
```

```
1    0.522963
0    0.477037
Name: fake, dtype: float64
```

## TextVectorization

Preprocess text and then map words to integers: we would create a frequency dictionary that encodes words with their total numbers of appearances in the dataset. And we set a limit of 2000 to only use the most frequent 2000 words to set up our word dictionary for training.

```
#preparing a title vectorization layer for tf model
size_vocabulary = 2000

#convert all strings to lower cases,
#get rid of all puncuations
def standardization(input_data):
    lowercase = tf.strings.lower(input_data)
    no_punctuation = tf.strings.regex_replace(lowercase,
                                  '[%s]' % re.escape(string.punctuation),'')
    return no_punctuation

title_vectorize_layer = layers.TextVectorization(
    standardize=standardization,
```

```
    max_tokens=size_vocabulary, # only consider this many words
    output_mode='int',
    output_sequence_length=500)

#this will make the layer 'learn' whatever words we've included from the titles
title_vectorize_layer.adapt(train_ds.map(lambda x, y: x["title"]))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.9/dist-packages/tensorflow/python/autograph/pyct/static_analysis/liveness.py:83: Analyzer.lamba_check (from tensorflow.python.autograph.pyct.static_analysis.liveness) is deprecated and will be removed after 2023-09-23.
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they are used, or at least in the same block. https://github.com/tensorflow/tensorflow/issues/56089
```

```
#preparing a text vectorization layer for tf model

def standardization(input_data):
    lowercase = tf.strings.lower(input_data)
    no_punctuation = tf.strings.regex_replace(lowercase,
                                  '[%s]' % re.escape(string.punctuation),'')
    return no_punctuation

text_vectorize_layer = layers.TextVectorization(
    standardize=standardization,
    max_tokens=size_vocabulary, # only consider this many words
    output_mode='int',
    output_sequence_length=500)

text_vectorize_layer.adapt(train_ds.map(lambda x, y: x["text"]))
```

## Create a Model

We would be building three models (using the functional API of keras) that train on only title, only text, both title and text respectively to answer the question:

**When detecting fake news, is it most effective to focus on only the title of the article, the full text of the article, or both?**

- In the first model, you should use only the article title as an input.
- In the second model, you should use only the article text as an input.
- In the third model, you should use both the article title and the article text as input.

(Applied to text vectorization layer adaptation as well)

As suggested, we define an embedding layer that would be shared by all three models.

```
max_tokens = 2000
output_sequence_length = 25
emb = layers.Embedding(max_tokens, output_dim = 3, name="embedding")
```

## Article title only

```
import keras
```

```
title_in = keras.Input(shape=(1,),name = "title", dtype = "string")
title_layer = title_vectorize_layer(title_in) #vectorize title
title_layer = emb(title_layer) #shared embedding
title_layer = layers.Dropout(0.2)(title_layer) #randomaly drop 20% of the connections to reduce overfitting
title_layer = layers.GlobalAveragePooling1D()(title_layer) #take the average of embedding vectors along the time axis
title_layer = layers.Dropout(0.2)(title_layer)
title_layer = layers.Dense(32, activation='relu')(title_layer)


# output layer
output = layers.Dense(2, name = "fake")(title_layer)
model1 = keras.Model(inputs = title_in,outputs = output,name='title_only')
model1.summary()
```

```
Model: "title_only"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 title (InputLayer)          [(None, 1)]               0

 text_vectorization (TextVec  (None, 500)              0
 torization)

 embedding (Embedding)       (None, 500, 3)            6000

 dropout (Dropout)           (None, 500, 3)            0

 global_average_pooling1d (G  (None, 3)                0
 lobalAveragePooling1D)

 dropout_1 (Dropout)         (None, 3)                 0

 dense (Dense)               (None, 32)                128

 fake (Dense)                (None, 2)                 66

=================================================================
Total params: 6,194
Trainable params: 6,194
Non-trainable params: 0
_________________________________________________________________
```

```
model1.compile(optimizer="adam",
              loss = losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=["accuracy"])
```

```
history1 = model1.fit(train_ds, epochs=20, validation_data=val_ds)
```

```
Epoch 1/20

/usr/local/lib/python3.9/dist-packages/keras/engine/functional.py:638: UserWarning: Input dict contained keys ['text']
which did not match any model input. They will be ignored by the model.
  inputs = self._flatten_to_reference_inputs(inputs)
```

```
180/180 [==============================] - 5s 19ms/step - loss: 0.6918 - accuracy: 0.5204 - val_loss: 0.6895 -
val_accuracy: 0.5248
Epoch 2/20
180/180 [==============================] - 3s 14ms/step - loss: 0.6809 - accuracy: 0.5791 - val_loss: 0.6627 -
val_accuracy: 0.6393
Epoch 3/20
180/180 [==============================] - 3s 18ms/step - loss: 0.6163 - accuracy: 0.7934 - val_loss: 0.5440 -
val_accuracy: 0.9353
Epoch 4/20
180/180 [==============================] - 6s 34ms/step - loss: 0.4662 - accuracy: 0.9038 - val_loss: 0.3705 -
val_accuracy: 0.9411
Epoch 5/20
180/180 [==============================] - 2s 12ms/step - loss: 0.3244 - accuracy: 0.9332 - val_loss: 0.2537 -
val_accuracy: 0.9504
Epoch 6/20
180/180 [==============================] - 2s 12ms/step - loss: 0.2380 - accuracy: 0.9452 - val_loss: 0.1815 -
val_accuracy: 0.9618
Epoch 7/20
180/180 [==============================] - 3s 16ms/step - loss: 0.1870 - accuracy: 0.9529 - val_loss: 0.1344 -
val_accuracy: 0.9711
Epoch 8/20
180/180 [==============================] - 2s 11ms/step - loss: 0.1579 - accuracy: 0.9562 - val_loss: 0.1206 -
val_accuracy: 0.9674
Epoch 9/20
180/180 [==============================] - 2s 11ms/step - loss: 0.1364 - accuracy: 0.9608 - val_loss: 0.0928 -
val_accuracy: 0.9762
Epoch 10/20
180/180 [==============================] - 2s 12ms/step - loss: 0.1171 - accuracy: 0.9671 - val_loss: 0.0967 -
val_accuracy: 0.9667
Epoch 11/20
180/180 [==============================] - 2s 12ms/step - loss: 0.1102 - accuracy: 0.9659 - val_loss: 0.0836 -
val_accuracy: 0.9718
Epoch 12/20
180/180 [==============================] - 3s 18ms/step - loss: 0.1007 - accuracy: 0.9684 - val_loss: 0.0698 -
val_accuracy: 0.9813
Epoch 13/20
180/180 [==============================] - 2s 11ms/step - loss: 0.0932 - accuracy: 0.9702 - val_loss: 0.0697 -
val_accuracy: 0.9818
```

```
Epoch 14/20
180/180 [==============================] - 2s 12ms/step - loss: 0.0884 - accuracy: 0.9715 - val_loss: 0.0608 -
val_accuracy: 0.9831
Epoch 15/20
180/180 [==============================] - 2s 11ms/step - loss: 0.0850 - accuracy: 0.9706 - val_loss: 0.0641 -
val_accuracy: 0.9827
Epoch 16/20
180/180 [==============================] - 3s 18ms/step - loss: 0.0792 - accuracy: 0.9737 - val_loss: 0.0552 -
val_accuracy: 0.9827
Epoch 17/20
180/180 [==============================] - 2s 12ms/step - loss: 0.0746 - accuracy: 0.9747 - val_loss: 0.0609 -
val_accuracy: 0.9782
Epoch 18/20
180/180 [==============================] - 2s 12ms/step - loss: 0.0756 - accuracy: 0.9738 - val_loss: 0.0476 -
val_accuracy: 0.9824
Epoch 19/20
180/180 [==============================] - 2s 12ms/step - loss: 0.0690 - accuracy: 0.9770 - val_loss: 0.0466 -
val_accuracy: 0.9842
Epoch 20/20
180/180 [==============================] - 2s 11ms/step - loss: 0.0682 - accuracy: 0.9753 - val_loss: 0.0424 -
val_accuracy: 0.9860
```

From the model fitting history, the title-input only fake news classification reaches a **98.60%** validation accuracy, which is slightly higher than the training accuracy **97.53%**.

## Article text only.

```python
text_in = keras.Input(shape=(1,),name = "text", dtype = "string")
text_layer = text_vectorize_layer(text_in) #vectorize title
text_layer = emb(text_layer) #shared embedding
text_layer = layers.Dropout(0.2)(text_layer) #randomaly drop 20% of the connections to reduce overfitting
text_layer = layers.GlobalAveragePooling1D()(text_layer) #take the average of embedding vectors along the time axis
text_layer = layers.Dropout(0.2)(text_layer)
text_layer = layers.Dense(32, activation='relu')(text_layer)


# output layer
```

```python
output = layers.Dense(2, name = "fake")(text_layer)
model2 = keras.Model(inputs = text_in,outputs = output,name='text_only')
model2.summary()
```

```
Model: "text_only"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 text (InputLayer)           [(None, 1)]               0         
                                                                 
 text_vectorization_1 (TextV  (None, 500)              0         
 ectorization)                                                   
                                                                 
 embedding (Embedding)       (None, 500, 3)            6000      
                                                                 
 dropout_2 (Dropout)         (None, 500, 3)            0         
                                                                 
 global_average_pooling1d_1   (None, 3)                0         
 (GlobalAveragePooling1D)                                        
                                                                 
 dropout_3 (Dropout)         (None, 3)                 0         
                                                                 
 dense_1 (Dense)             (None, 32)                128       
                                                                 
 fake (Dense)                (None, 2)                 66        
                                                                 
=================================================================
Total params: 6,194
Trainable params: 6,194
Non-trainable params: 0
_________________________________________________________________
```

```python
model2.compile(optimizer="adam",
              loss = losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=["accuracy"])
```

```python
history2 = model2.fit(train_ds, epochs=20, validation_data=val_ds)
```

```
Epoch 1/20

/usr/local/lib/python3.9/dist-packages/keras/engine/functional.py:638: UserWarning: Input dict contained keys ['title']
which did not match any model input. They will be ignored by the model.
  inputs = self._flatten_to_reference_inputs(inputs)

180/180 [==============================] - 5s 20ms/step - loss: 0.6887 - accuracy: 0.5390 - val_loss: 0.6803 -
val_accuracy: 0.5831
Epoch 2/20
180/180 [==============================] - 4s 20ms/step - loss: 0.6538 - accuracy: 0.6232 - val_loss: 0.5856 -
val_accuracy: 0.8478
Epoch 3/20
180/180 [==============================] - 4s 23ms/step - loss: 0.5290 - accuracy: 0.7648 - val_loss: 0.3800 -
val_accuracy: 0.8930
Epoch 4/20
180/180 [==============================] - 4s 20ms/step - loss: 0.3859 - accuracy: 0.8456 - val_loss: 0.2583 -
val_accuracy: 0.9236
Epoch 5/20
180/180 [==============================] - 5s 27ms/step - loss: 0.3007 - accuracy: 0.8802 - val_loss: 0.2244 -
val_accuracy: 0.9271
Epoch 6/20
180/180 [==============================] - 4s 20ms/step - loss: 0.2516 - accuracy: 0.9063 - val_loss: 0.1965 -
val_accuracy: 0.9413
Epoch 7/20
180/180 [==============================] - 4s 20ms/step - loss: 0.2223 - accuracy: 0.9219 - val_loss: 0.1795 -
val_accuracy: 0.9476
Epoch 8/20
180/180 [==============================] - 5s 26ms/step - loss: 0.1967 - accuracy: 0.9338 - val_loss: 0.1423 -
val_accuracy: 0.9580
Epoch 9/20
180/180 [==============================] - 4s 20ms/step - loss: 0.1759 - accuracy: 0.9415 - val_loss: 0.1314 -
val_accuracy: 0.9611
Epoch 10/20
180/180 [==============================] - 4s 20ms/step - loss: 0.1654 - accuracy: 0.9447 - val_loss: 0.1214 -
val_accuracy: 0.9667
Epoch 11/20
180/180 [==============================] - 5s 27ms/step - loss: 0.1584 - accuracy: 0.9474 - val_loss: 0.1189 -
val_accuracy: 0.9682
```

```
Epoch 12/20
180/180 [==============================] - 3s 19ms/step - loss: 0.1469 - accuracy: 0.9519 - val_loss: 0.0997 -
val_accuracy: 0.9719
Epoch 13/20
180/180 [==============================] - 4s 24ms/step - loss: 0.1371 - accuracy: 0.9566 - val_loss: 0.1065 -
val_accuracy: 0.9707
Epoch 14/20
180/180 [==============================] - 4s 20ms/step - loss: 0.1400 - accuracy: 0.9560 - val_loss: 0.1080 -
val_accuracy: 0.9709
Epoch 15/20
180/180 [==============================] - 3s 19ms/step - loss: 0.1325 - accuracy: 0.9571 - val_loss: 0.0907 -
val_accuracy: 0.9747
Epoch 16/20
180/180 [==============================] - 4s 24ms/step - loss: 0.1227 - accuracy: 0.9605 - val_loss: 0.0901 -
val_accuracy: 0.9731
Epoch 17/20
180/180 [==============================] - 4s 20ms/step - loss: 0.1206 - accuracy: 0.9610 - val_loss: 0.0938 -
val_accuracy: 0.9749
Epoch 18/20
180/180 [==============================] - 3s 19ms/step - loss: 0.1154 - accuracy: 0.9621 - val_loss: 0.0848 -
val_accuracy: 0.9762
Epoch 19/20
180/180 [==============================] - 4s 23ms/step - loss: 0.1116 - accuracy: 0.9645 - val_loss: 0.0798 -
val_accuracy: 0.9782
Epoch 20/20
180/180 [==============================] - 4s 20ms/step - loss: 0.1072 - accuracy: 0.9649 - val_loss: 0.0790 -
val_accuracy: 0.9800
```

From the model fitting history, the title-input only fake news classification reaches a **98.00%** validation accuracy, which is slightly higher than the training accuracy **96.49%**, performing a little bit worse than the title-only model.

## Combining text and title

We use [concatenate](https://keras.io) to combine the above two model

```
both = layers.concatenate([title_layer, text_layer], axis=1)
both = layers.Dense(32, activation='relu')(both)
```

```
output = layers.Dense(2, name = "fake")(both)
model3 = keras.Model(inputs = [title_in,text_in],outputs = output,name='both')
model3.summary()
```

```
Model: "both"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to                     
==================================================================================================
 title (InputLayer)             [(None, 1)]          0           []                               
                                                                                                  
 text (InputLayer)              [(None, 1)]          0           []                               
                                                                                                  
 text_vectorization (TextVector  (None, 500)         0           ['title[0][0]']                  
 ization)                                                                                         
                                                                                                  
 text_vectorization_1 (TextVect  (None, 500)         0           ['text[0][0]']                   
 orization)                                                                                       
                                                                                                  
 embedding (Embedding)          (None, 500, 3)       6000        ['text_vectorization[0][0]',     
                                                                  'text_vectorization_1[0][0]']   
                                                                                                  
 dropout (Dropout)              (None, 500, 3)       0           ['embedding[0][0]']              
                                                                                                  
 dropout_2 (Dropout)            (None, 500, 3)       0           ['embedding[1][0]']              
                                                                                                  
 global_average_pooling1d (Glob  (None, 3)           0           ['dropout[0][0]']                
 alAveragePooling1D)                                                                              
                                                                                                  
 global_average_pooling1d_1 (Gl  (None, 3)           0           ['dropout_2[0][0]']              
 obalAveragePooling1D)                                                                            
                                                                                                  
 dropout_1 (Dropout)            (None, 3)            0           ['global_average_pooling1d[0][0]'
                                                                 ]                                
                                                                                                  
 dropout_3 (Dropout)            (None, 3)            0           ['global_average_pooling1d_1[0][0
                                                                 ]']                              
                                                                                                  
 dense (Dense)                  (None, 32)           128         ['dropout_1[0][0]']              
```

```
 dense_1 (Dense)                (None, 32)           128         ['dropout_3[0][0]']

 concatenate (Concatenate)      (None, 64)           0           ['dense[0][0]',
                                                                  'dense_1[0][0]']

 dense_2 (Dense)                (None, 32)           2080        ['concatenate[0][0]']

 fake (Dense)                   (None, 2)            66          ['dense_2[0][0]']

==================================================================================================
Total params: 8,402
Trainable params: 8,402
Non-trainable params: 0
__________________________________________________________________________________________________
```

```python
model3.compile(optimizer="adam",
              loss = losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=["accuracy"])
```

```python
history3 = model3.fit(train_ds, epochs=20, validation_data=val_ds)
```

```
Epoch 1/20
180/180 [==============================] - 5s 21ms/step - loss: 0.6377 - accuracy: 0.6446 - val_loss: 0.4093 -
val_accuracy: 0.9131
Epoch 2/20
180/180 [==============================] - 5s 25ms/step - loss: 0.2503 - accuracy: 0.9271 - val_loss: 0.1526 -
val_accuracy: 0.9618
Epoch 3/20
180/180 [==============================] - 4s 19ms/step - loss: 0.1470 - accuracy: 0.9570 - val_loss: 0.1071 -
val_accuracy: 0.9707
Epoch 4/20
180/180 [==============================] - 5s 26ms/step - loss: 0.1141 - accuracy: 0.9665 - val_loss: 0.0888 -
val_accuracy: 0.9776
Epoch 5/20
180/180 [==============================] - 4s 20ms/step - loss: 0.0986 - accuracy: 0.9694 - val_loss: 0.0654 -
val_accuracy: 0.9834
```

```
Epoch 6/20
180/180 [==============================] - 4s 20ms/step - loss: 0.0909 - accuracy: 0.9730 - val_loss: 0.0627 -
val_accuracy: 0.9867
Epoch 7/20
180/180 [==============================] - 4s 20ms/step - loss: 0.0773 - accuracy: 0.9765 - val_loss: 0.0599 -
val_accuracy: 0.9864
Epoch 8/20
180/180 [==============================] - 4s 21ms/step - loss: 0.0756 - accuracy: 0.9770 - val_loss: 0.0550 -
val_accuracy: 0.9858
Epoch 9/20
180/180 [==============================] - 4s 23ms/step - loss: 0.0661 - accuracy: 0.9814 - val_loss: 0.0408 -
val_accuracy: 0.9904
Epoch 10/20
180/180 [==============================] - 4s 21ms/step - loss: 0.0573 - accuracy: 0.9818 - val_loss: 0.0403 -
val_accuracy: 0.9904
Epoch 11/20
180/180 [==============================] - 4s 24ms/step - loss: 0.0586 - accuracy: 0.9822 - val_loss: 0.0433 -
val_accuracy: 0.9911
Epoch 12/20
180/180 [==============================] - 4s 21ms/step - loss: 0.0581 - accuracy: 0.9823 - val_loss: 0.0336 -
val_accuracy: 0.9933
Epoch 13/20
180/180 [==============================] - 4s 21ms/step - loss: 0.0499 - accuracy: 0.9838 - val_loss: 0.0409 -
val_accuracy: 0.9911
Epoch 14/20
180/180 [==============================] - 4s 23ms/step - loss: 0.0509 - accuracy: 0.9845 - val_loss: 0.0336 -
val_accuracy: 0.9916
Epoch 15/20
180/180 [==============================] - 4s 21ms/step - loss: 0.0468 - accuracy: 0.9847 - val_loss: 0.0279 -
val_accuracy: 0.9929
Epoch 16/20
180/180 [==============================] - 5s 27ms/step - loss: 0.0450 - accuracy: 0.9852 - val_loss: 0.0284 -
val_accuracy: 0.9922
Epoch 17/20
180/180 [==============================] - 4s 20ms/step - loss: 0.0429 - accuracy: 0.9850 - val_loss: 0.0246 -
val_accuracy: 0.9936
Epoch 18/20
180/180 [==============================] - 5s 26ms/step - loss: 0.0395 - accuracy: 0.9869 - val_loss: 0.0253 -
```

```
val_accuracy: 0.9940
Epoch 19/20
180/180 [==============================] - 4s 21ms/step - loss: 0.0426 - accuracy: 0.9845 - val_loss: 0.0236 -
val_accuracy: 0.9955
Epoch 20/20
180/180 [==============================] - 4s 21ms/step - loss: 0.0407 - accuracy: 0.9864 - val_loss: 0.0206 -
val_accuracy: 0.9964
```

We reached the highest validation accuracy so far - **99.64%** with a training accuracy of **98.64%** using both text and title.

## Model Evaluation.

Let's examine how well our classification model performs on unforseen data.

```
test_url = "https://github.com/PhilChodrow/PIC16b/blob/master/datasets/fake_news_test.csv?raw=true"
```

```
testdf = pd.read_csv(test_url)
```

```
test = make_dataset(testdf)
```

```
model3.metrics_names
```

```
[]
```

```
model3.evaluate(test)
```

```
225/225 [==============================] - 2s 8ms/step - loss: 0.0765 - accuracy: 0.9815
```

```
[0.07650567591190338, 0.9815136790275574]
```

We achieved a **98.15% accuracy** in fake news classification on the test data.

# Embedding Visualization

A word embedding is a learned representation for text where words that have the same meaning have a similar representation. One of the ways to learn word embedding is through an embedding layer, a word embedding that is learned jointly with a neural network model on a specific natural language processing task, such as fake news classification.

We will use PCA (principal component analysis) to distill the embedding down to two dimensions for ease of visualization while perserving the variations among words.

```
text_vectorize_layer.adapt(train_ds.map(lambda x, y: x["title"]))
```

```
vocab = text_vectorize_layer.get_vocabulary() # keeps track of mapping from word to integer
```

```
weights = model3.get_layer("embedding").get_weights()[0]
```

```
weights.shape # 2000 vocabs x 3 dimensional space
```

```
(2000, 3)
```

```
from sklearn.decomposition import PCA
# https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
# principal components analysis -
# project things to lower dimension such that the variance of the dataset is most preserved

pca = PCA(n_components=2)
weights = pca.fit_transform(weights)
```

```
embedding_df = pd.DataFrame({
    'word': vocab,
    'x0': weights[:, 0],
```

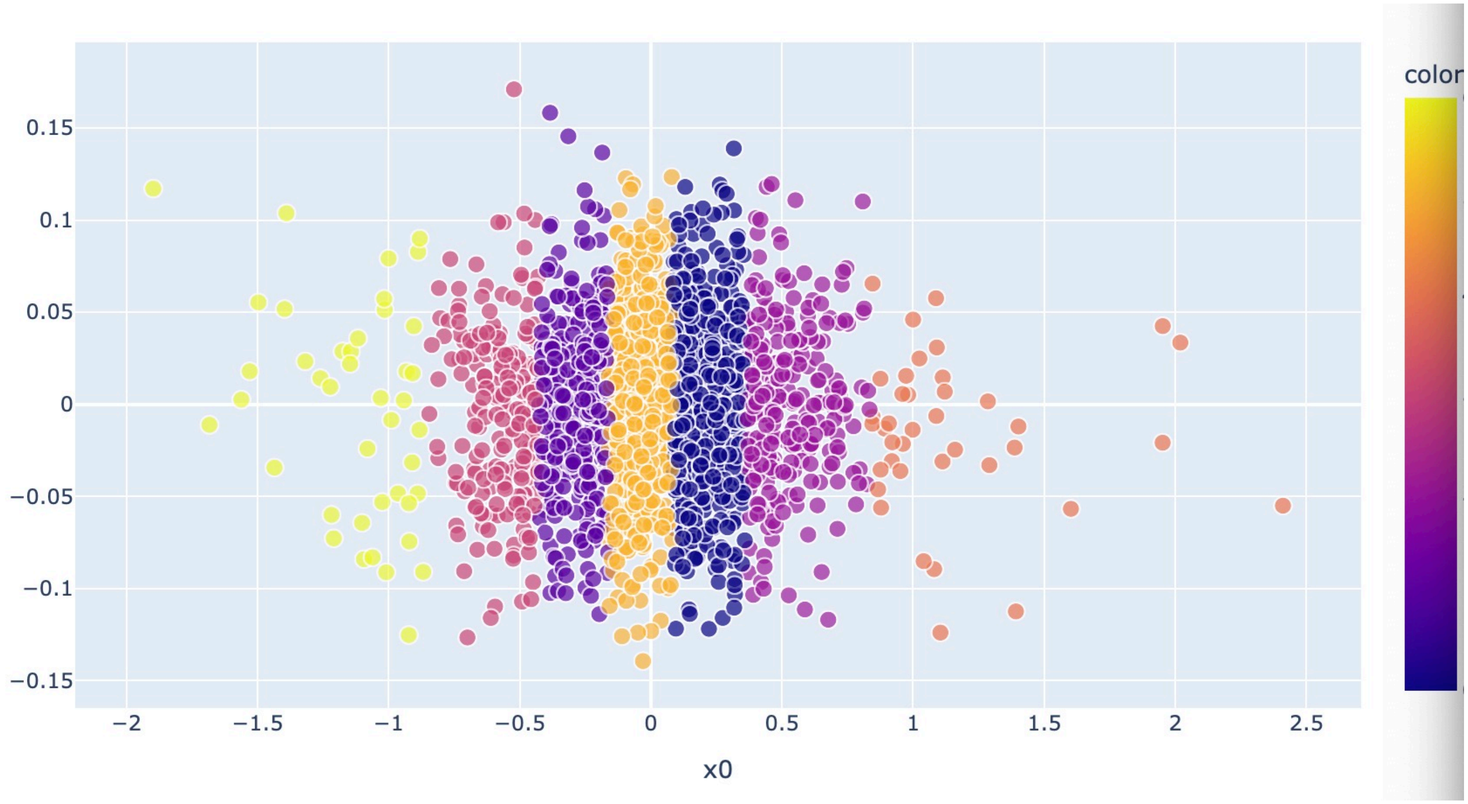
```
    'x1': weights[:, 1]
})
```

We proceed to color the embedding [KMeans(https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)]

```
from sklearn.cluster import KMeans
import numpy as np
X = embedding_df[['x0','x1']]
kmeans = KMeans(n_clusters=7, random_state=0, n_init="auto").fit(X)
embedding_df['color'] = kmeans.labels_
kmeans.cluster_centers_
```

```
array([[ 2.0610277e-01,  1.6437012e-03],
       [-2.8306237e-01, -3.1798152e-04],
       [ 5.2419913e-01, -1.2679024e-04],
       [-5.6891716e-01, -5.1641576e-03],
       [ 1.1591365e+00, -1.5719092e-02],
       [-3.8863741e-02,  2.0008855e-03],
       [-1.1274347e+00, -2.8834003e-03]], dtype=float32)
```

```
import plotly.express as px
fig = px.scatter(embedding_df,
                 x='x0',
                 y='x1',
                 size=[2]*len(embedding_df),
                 size_max = 10,
                 hover_name = 'word',
                 color='color'
                 )
fig.show()
fig
```




1. The clusterings simply divide the words based on their x0 weights, so the variations mainly exist there. Some of the significant outliers of the x0 axis are **Trumps** (Trump's), rep, gop, j, im, more, mr, that, nov. Some of them are more like stop words (I'm, that before standardization) that appear a lot but don't really mean anything. Trump is an outlier because obviously he's one of his kind in terms of spreading false information, making false claims, and creating chaos on the social network (Trump's twitter).
2. Meanwhile, **trump** appears somewhere in the middle orange cluster as well as some other politician last name (like **clinton**). The standardization does not collapse trumps and trump into the same thing. So politician last names mostly appear in the same group.
3. In terms of the x1 axis, one word that gets weighted heavily is **knowledge**. It's quite easy to think of sentences like 'currently, scientists have xx knowledge on...', 'officials claimed no knowledge of...' to be in a supposedly 'informative' piece of news.
4. The rightest cluster has words like **friday, tuesday, thurdsay** that belong to the same category. To the right, there are adverbs like **allegedly, apparently, recently** that seems to be common in all news articles.
5. The middle clusters are mostly noun, proper nouns, and verb that are not weighted heavily.