fake news classification

In this blog post, we will begin a tutorial in Tensorflow related to text classification. Here,we are going to develop a fake news classification to determine the increasing number of fake news in our life.

Before we start coding, we will need to import some necessary packages

```
import numpy as np
import pandas as pd
import tensorflow as tf
import re
import string

from tensorflow.keras import layers
from tensorflow.keras import losses
from tensorflow import keras
from nltk.corpus import stopwords

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras import utils

# for embedding viz
import plotly.express as px
import plotly.io as pio
import matplotlib.pyplot as plt
pio.templates.default = "plotly_white"
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
from sklearn.decomposition import PCA
```

Part1:Acquire Training Data

Our trainning dataset comes from the below “train_url”,we use pd.read_csv() to directly read it into Python.

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

	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
...	...	...	...	...
22444	10709	ALARMING: NSA Refuses to Release Clinton-Lynch...	If Clinton and Lynch just talked about grandki...	1
22445	8731	Can Pence's vow not to sling mud survive a Tru...	() - In 1990, during a close and bitter congre...	0
22446	4733	Watch Trump Campaign Try To Spin Their Way Ou...	A new ad by the Hillary Clinton SuperPac Prior...	1
22447	3993	Trump celebrates first 100 days as president, ...	HARRISBURG, Pa.U.S. President Donald Trump hit...	0
22448	12896	TRUMP SUPPORTERS REACT TO DEBATE: “Clinton New...	MELBOURNE, FL is a town with a population of 7...	1

22449 rows × 4 columns

Part2:Make a Dataset

In this part,we will begin our basic step to prepare for training model–construct our database.

Step 1:Define make_dataset() function

Frist, we need to define a function,which named make_dataset().The functio has two main roles:

- Remove stopwords from the article text and title.
- Construct and return a tf.data.Dataset with two inputs and one output.

> Input is of the form (title, text), and the output is consist only of the fake column.

```
def make_dataset(df):
  #import stopwords
  import nltk
  nltk.download('stopwords')
  stop = stopwords.words('english')
  #remove stopwords from title and text
  df['title'] = df['title'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
  df['text'] = df['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
  data = tf.data.Dataset.from_tensor_slices(
   ( # dictionary for input data
       {
        "title": df[["title"]], # gives a dataframe
        "text": df[["text"]]
    },
    # dictionary for output data
    {
        "fake": df[["fake"]]
    }
   ) )

  return data.batch(100)
```

```
data=make_dataset(df)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/a10033/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Step 2:Validation Data

After we've constructed our primary Dataset, split of 20% of it to use for validation.

```
data = data.shuffle(buffer_size = len(data))
#80% for train,20% for validation
train_size = int(0.8*len(data))
val_size   = int(0.2*len(data))

train = data.take(train_size)# data[:train_size]
val = data.skip(train_size).take(val_size)# data[train_size : train_size + val_size]
```

According to the output,we can check that we split 80% for the training dataset and 20% for the validation dataset.

```
print(len(train), len(val))
```

```
180 45
```

Step 3:Base Rate

Since the base rate refers to the accuracy of a model that always makes the same guess.We will get the guess rate in this step by examining the lables on the training set.

- Fake News(label 0)
- Real News(label 1)

```
#numpy.ndarray
labels_iterator= train.unbatch().map(lambda image, label: label["fake"]).as_numpy_iterator()
#convert to list
labels_list=list(labels_iterator)
#count the frequency of fake news and real news
total=len(labels_list)
fake_f=labels_list.count(1)
real_f=labels_list.count(0)
print("Number of total news in the traing data:"+str(total))
```

```
print("Number of fake news in the training data:"+str(fake_f))
print("Number of real news in the training data:"+str(real_f))
```

```
Number of total news in the traing data:17949
Number of fake news in the training data:9420
Number of real news in the training data:8529
```

Based on the counted number, we can find that the base rate is 52%.Since our aim is to check the fake news,we use the number of fake news to calculate the base rate.

Step 4:TextVectorization

In this step,we created the title_vectorize_layer and text_vectorize_layer.

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

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 = TextVectorization(
    standardize=standardization,
    max_tokens=size_vocabulary, # only consider this many words
    output_mode='int',#get frequency ranking
    output_sequence_length=500)
text_vectorize_layer = TextVectorization(
    standardize=standardization,
    max_tokens=size_vocabulary, # only consider this many words
    output_mode='int',#get frequency ranking
    output_sequence_length=500)
title_vectorize_layer.adapt(train.map(lambda x, y: x["title"]))

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

Part3: Create Models

Now we encourntered a problem here: 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?

To solve this problem, we will develop three models:

- Model1:only the article title as an input.
- Model2:only the article text as an input.
- Model3:both the article title and the article text as input.

Also,We can create a plot_accuracy() function to help us visualize the accuracy of our model.The x axis is the epoch and y axis is accuracy. In our graph, we can observe validation accuracy and training accuracy.

```
def plot_accuracy(history):
  acc = history.history['accuracy']
  val_acc = history.history['val_accuracy']
  plt.plot(acc, label='Training Accuracy')
  plt.plot(val_acc, label='Validation Accuracy')
  plt.title('Training and Validation Accuracy')
  plt.xlabel('epoch')
  plt.ylabel('Accuracy')
  plt.legend()
  plt.show()
```

In addition to the plot_accuracy function, we need to specify two inputs and one sharing embedding.

- title input
- text input
- shared embedding

```
title_input = keras.Input(
    shape=(1,),
    name = "title", # same name as the dictionary key in the dataset
```

```
    dtype = "string"
)

text_input = keras.Input(
    shape=(1, ),
    name = "text",
    dtype = "string"
)
```

Here we created an embedding in a relatively large number of dimensions (10) and then use PCA to reduce the dimension down to a visualizable number.

```
shared_embedding = layers.Embedding(size_vocabulary, output_dim = 10, name="embedding")
```

Model 1: Article Title

constructing layers

- shared_embedding
- title_vectorize_layer
- dropout
- dense
- GlobalAveragePooling1D

```
title_features = title_vectorize_layer(title_input) # apply this "function TextVectorization layer" to title_input
title_features = shared_embedding(title_features)
title_features = layers.Dropout(0.2)(title_features)
title_features = layers.GlobalAveragePooling1D()(title_features)
title_features = layers.Dropout(0.2)(title_features)
title_features = layers.Dense(32, activation='relu')(title_features)
```

output layer

```
output1 = layers.Dense(2, name="fake")(title_features)
```

Complie the Model

```
#the input is only title
model1 = keras.Model(
    inputs = [title_input],
    outputs = output1
)
model1.compile(optimizer="adam",
              loss = losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=["accuracy"])
```

Train the Model

```
history1 = model1.fit(train,
                    validation_data=val,
                    epochs = 20)
```

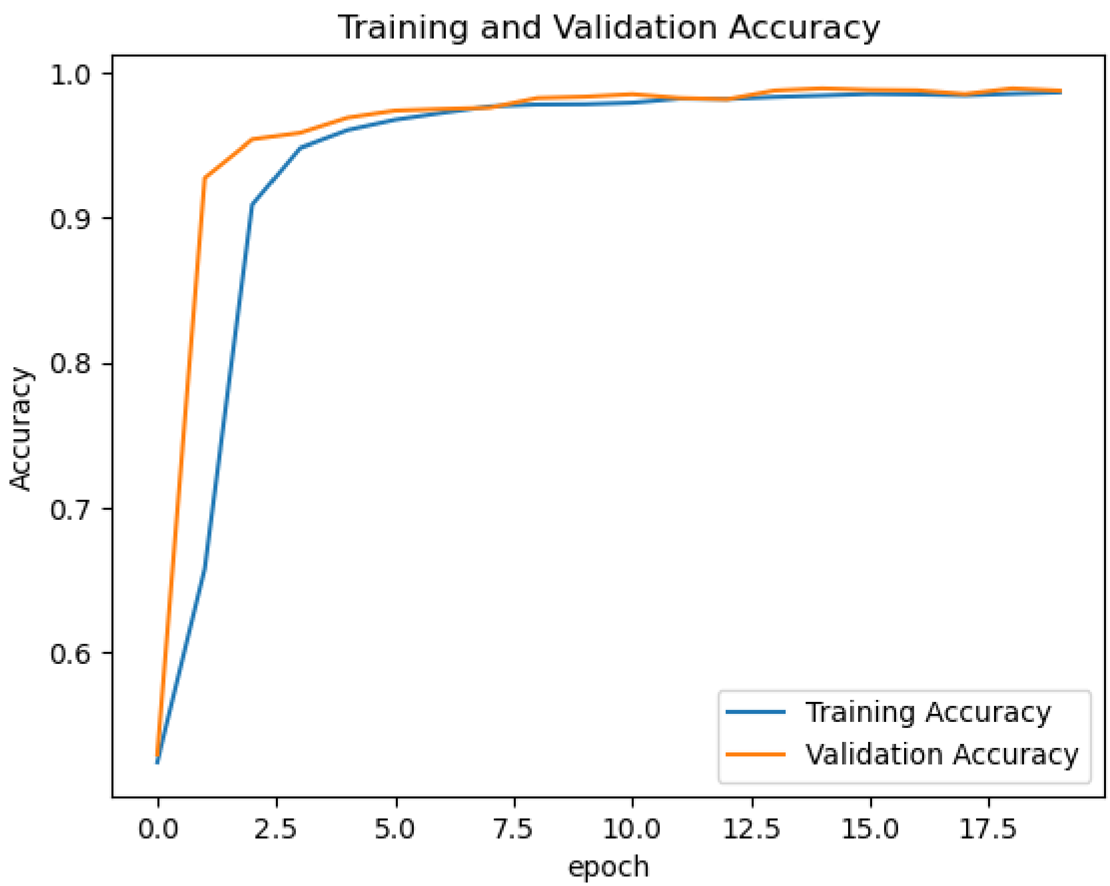
```
Epoch 1/20

/Users/a10033/opt/anaconda3/envs/PIC16B/lib/python3.8/site-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.

180/180 [==============================] - 2s 7ms/step - loss: 0.6906 - accuracy: 0.5248 - val_loss: 0.6853 -
val_accuracy: 0.5302
Epoch 2/20
180/180 [==============================] - 1s 7ms/step - loss: 0.6564 - accuracy: 0.6581 - val_loss: 0.5903 -
val_accuracy: 0.9274
Epoch 3/20
180/180 [==============================] - 1s 7ms/step - loss: 0.4600 - accuracy: 0.9090 - val_loss: 0.3203 -
val_accuracy: 0.9538
Epoch 4/20
180/180 [==============================] - 1s 6ms/step - loss: 0.2523 - accuracy: 0.9479 - val_loss: 0.1891 -
val_accuracy: 0.9582
Epoch 5/20
180/180 [==============================] - 1s 6ms/step - loss: 0.1650 - accuracy: 0.9601 - val_loss: 0.1310 -
val_accuracy: 0.9687
Epoch 6/20
```

```
180/180 [==============================] - 1s 6ms/step - loss: 0.1236 - accuracy: 0.9672 - val_loss: 0.1008 -
val_accuracy: 0.9736
Epoch 7/20
180/180 [==============================] - 1s 6ms/step - loss: 0.0994 - accuracy: 0.9720 - val_loss: 0.0909 -
val_accuracy: 0.9747
Epoch 8/20
180/180 [==============================] - 1s 6ms/step - loss: 0.0857 - accuracy: 0.9764 - val_loss: 0.0783 -
val_accuracy: 0.9756
Epoch 9/20
180/180 [==============================] - 1s 6ms/step - loss: 0.0754 - accuracy: 0.9778 - val_loss: 0.0658 -
val_accuracy: 0.9822
Epoch 10/20
180/180 [==============================] - 1s 6ms/step - loss: 0.0690 - accuracy: 0.9779 - val_loss: 0.0608 -
val_accuracy: 0.9831
Epoch 11/20
180/180 [==============================] - 1s 7ms/step - loss: 0.0656 - accuracy: 0.9790 - val_loss: 0.0520 -
val_accuracy: 0.9849
Epoch 12/20
180/180 [==============================] - 1s 7ms/step - loss: 0.0599 - accuracy: 0.9818 - val_loss: 0.0527 -
val_accuracy: 0.9822
Epoch 13/20
180/180 [==============================] - 1s 7ms/step - loss: 0.0583 - accuracy: 0.9815 - val_loss: 0.0518 -
val_accuracy: 0.9813
Epoch 14/20
180/180 [==============================] - 1s 7ms/step - loss: 0.0543 - accuracy: 0.9829 - val_loss: 0.0429 -
val_accuracy: 0.9874
Epoch 15/20
180/180 [==============================] - 1s 7ms/step - loss: 0.0503 - accuracy: 0.9840 - val_loss: 0.0419 -
val_accuracy: 0.9889
Epoch 16/20
180/180 [==============================] - 1s 7ms/step - loss: 0.0474 - accuracy: 0.9851 - val_loss: 0.0434 -
val_accuracy: 0.9879
Epoch 17/20
180/180 [==============================] - 1s 7ms/step - loss: 0.0473 - accuracy: 0.9847 - val_loss: 0.0388 -
val_accuracy: 0.9876
Epoch 18/20
180/180 [==============================] - 1s 7ms/step - loss: 0.0465 - accuracy: 0.9840 - val_loss: 0.0412 -
val_accuracy: 0.9849
```

```
Epoch 19/20
180/180 [==============================] - 1s 6ms/step - loss: 0.0437 - accuracy: 0.9852 - val_loss: 0.0347 -
val_accuracy: 0.9889
Epoch 20/20
180/180 [==============================] - 1s 6ms/step - loss: 0.0412 - accuracy: 0.9862 - val_loss: 0.0342 -
val_accuracy: 0.9873
```

Plot the Accuracy

```
plot_accuracy(history1)
```



Model 2: Article Text

constructing layers

- shared_embedding
- title_vectorize_layer
- dropout
- dense
- GlobalAveragePooling1D

```
text_features = text_vectorize_layer(text_input) # apply this "function TextVectorization layer" to text_input
text_features = shared_embedding(text_features)
text_features = layers.Dropout(0.2)(text_features)
text_features = layers.GlobalAveragePooling1D()(text_features)
text_features = layers.Dropout(0.2)(text_features)
text_features = layers.Dense(32, activation='relu')(text_features)
```

Output Layer

```
output2 = layers.Dense(2, name="fake")(text_features)
```

Complie the Model

```
#input is onlt text
model2 = keras.Model(
    inputs = [text_input],
    outputs = output2
)
model2.compile(optimizer="adam",
              loss = losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=["accuracy"])
```

Train the Model

```
history2 = model2.fit(train,
                    validation_data=val,
```

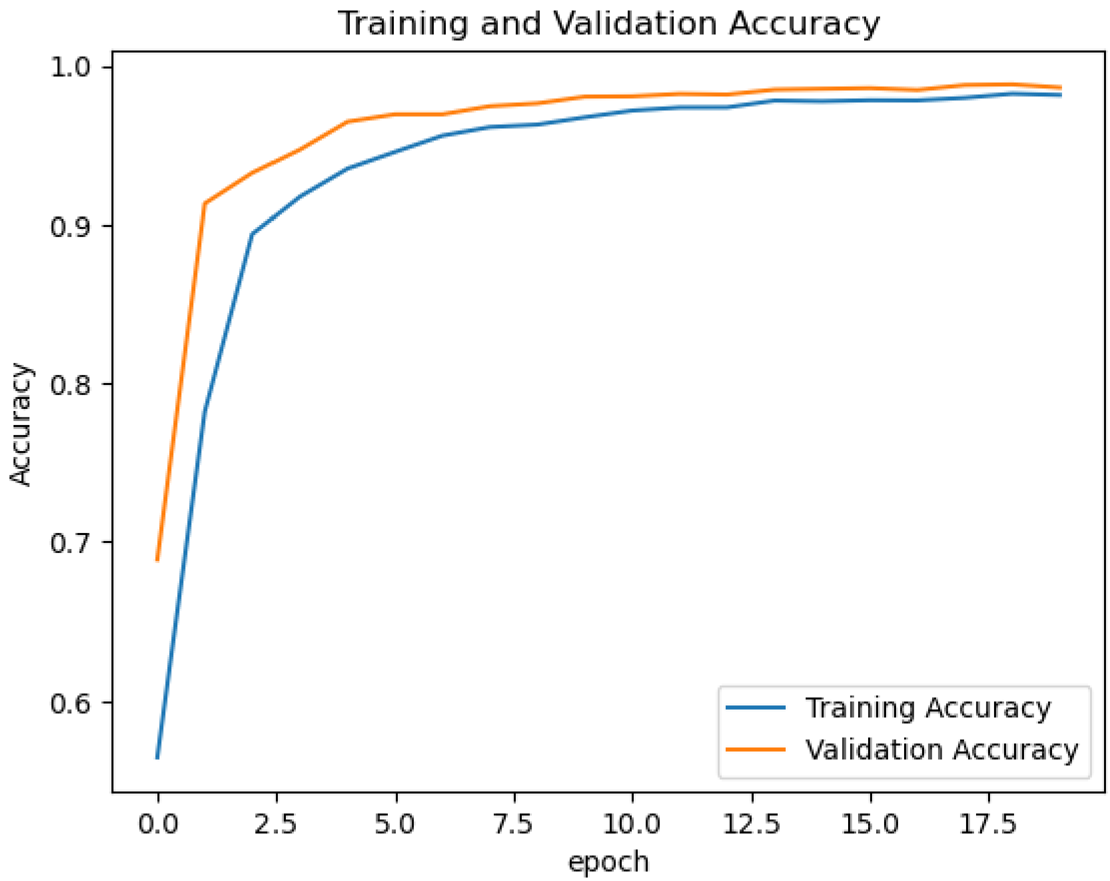
```
                    epochs = 20)
```

```
Epoch 1/20

/Users/a10033/opt/anaconda3/envs/PIC16B/lib/python3.8/site-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.

180/180 [==============================] - 3s 14ms/step - loss: 0.6814 - accuracy: 0.5645 - val_loss: 0.6450 -
val_accuracy: 0.6889
Epoch 2/20
180/180 [==============================] - 2s 13ms/step - loss: 0.5415 - accuracy: 0.7821 - val_loss: 0.3571 -
val_accuracy: 0.9131
Epoch 3/20
180/180 [==============================] - 2s 13ms/step - loss: 0.3185 - accuracy: 0.8941 - val_loss: 0.2246 -
val_accuracy: 0.9324
Epoch 4/20
180/180 [==============================] - 2s 13ms/step - loss: 0.2391 - accuracy: 0.9175 - val_loss: 0.1833 -
val_accuracy: 0.9471
Epoch 5/20
180/180 [==============================] - 2s 13ms/step - loss: 0.1934 - accuracy: 0.9351 - val_loss: 0.1487 -
val_accuracy: 0.9647
Epoch 6/20
180/180 [==============================] - 2s 13ms/step - loss: 0.1660 - accuracy: 0.9456 - val_loss: 0.1247 -
val_accuracy: 0.9693
Epoch 7/20
180/180 [==============================] - 2s 13ms/step - loss: 0.1385 - accuracy: 0.9559 - val_loss: 0.1065 -
val_accuracy: 0.9693
Epoch 8/20
180/180 [==============================] - 3s 14ms/step - loss: 0.1273 - accuracy: 0.9613 - val_loss: 0.1019 -
val_accuracy: 0.9744
Epoch 9/20
180/180 [==============================] - 3s 14ms/step - loss: 0.1161 - accuracy: 0.9628 - val_loss: 0.0958 -
val_accuracy: 0.9762
Epoch 10/20
180/180 [==============================] - 3s 15ms/step - loss: 0.1060 - accuracy: 0.9674 - val_loss: 0.0753 -
val_accuracy: 0.9804
Epoch 11/20
```

```
180/180 [==============================] - 2s 13ms/step - loss: 0.0954 - accuracy: 0.9717 - val_loss: 0.0739 -
val_accuracy: 0.9807
Epoch 12/20
180/180 [==============================] - 2s 13ms/step - loss: 0.0864 - accuracy: 0.9736 - val_loss: 0.0740 -
val_accuracy: 0.9822
Epoch 13/20
180/180 [==============================] - 2s 14ms/step - loss: 0.0872 - accuracy: 0.9737 - val_loss: 0.0659 -
val_accuracy: 0.9818
Epoch 14/20
180/180 [==============================] - 3s 14ms/step - loss: 0.0767 - accuracy: 0.9780 - val_loss: 0.0592 -
val_accuracy: 0.9849
Epoch 15/20
180/180 [==============================] - 3s 14ms/step - loss: 0.0751 - accuracy: 0.9775 - val_loss: 0.0588 -
val_accuracy: 0.9854
Epoch 16/20
180/180 [==============================] - 2s 13ms/step - loss: 0.0713 - accuracy: 0.9782 - val_loss: 0.0494 -
val_accuracy: 0.9858
Epoch 17/20
180/180 [==============================] - 2s 13ms/step - loss: 0.0717 - accuracy: 0.9781 - val_loss: 0.0554 -
val_accuracy: 0.9847
Epoch 18/20
180/180 [==============================] - 2s 13ms/step - loss: 0.0685 - accuracy: 0.9797 - val_loss: 0.0455 -
val_accuracy: 0.9878
Epoch 19/20
180/180 [==============================] - 2s 13ms/step - loss: 0.0602 - accuracy: 0.9823 - val_loss: 0.0487 -
val_accuracy: 0.9882
Epoch 20/20
180/180 [==============================] - 2s 13ms/step - loss: 0.0606 - accuracy: 0.9815 - val_loss: 0.0466 -
val_accuracy: 0.9862
```

Plot the Accuracy

```
plot_accuracy(history2)
```



Model 3:Article Title & Article Text

constructing layers

```
main = layers.concatenate([title_features, text_features], axis = 1)
main = layers.Dense(32, activation='relu')(main)
output3 = layers.Dense(2, name="fake")(main)
```

Complie the Model

```
#input are text and title
model3 = keras.Model(
    inputs = [title_input, text_input],
    outputs = output3
)
model3.compile(optimizer="adam",
              loss = losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=["accuracy"])
```

Train the Model

```
history3 = model3.fit(train,
                    validation_data=val,
                    epochs = 20)
```

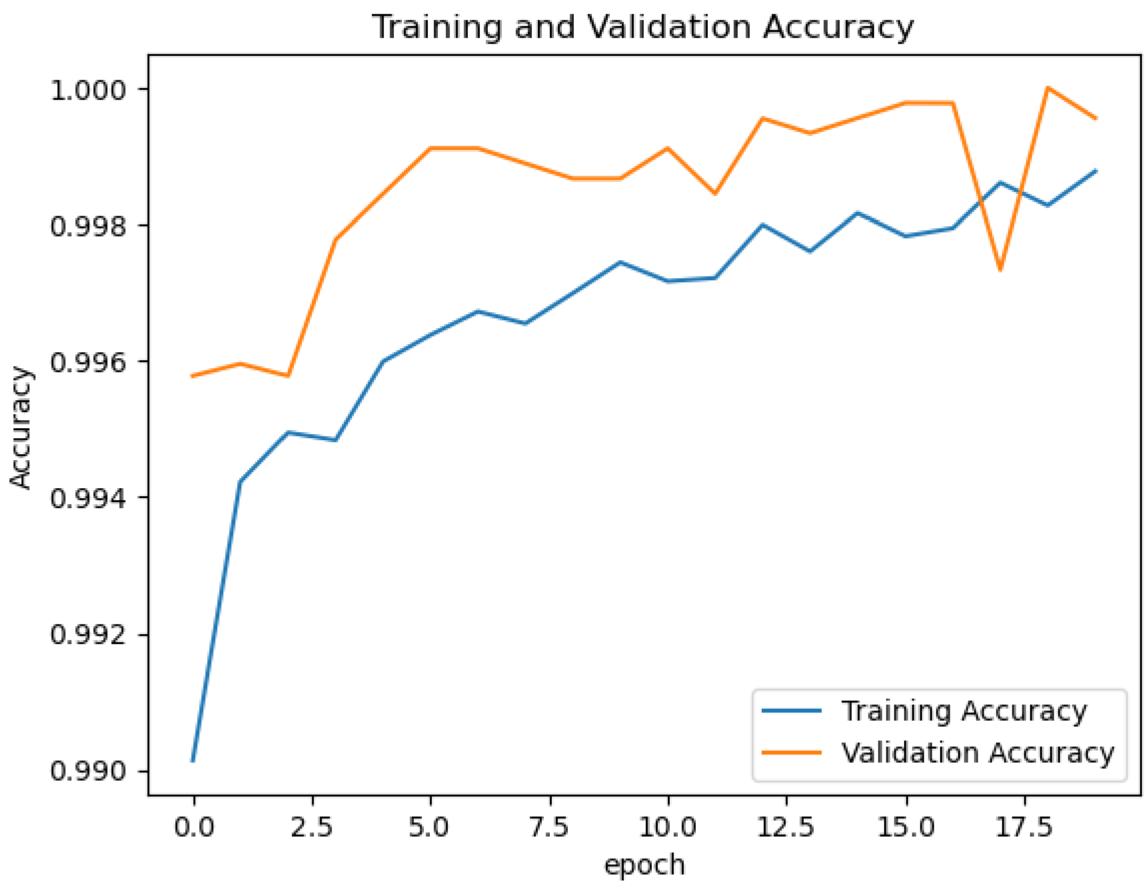
```
Epoch 1/20
180/180 [==============================] - 4s 18ms/step - loss: 0.1495 - accuracy: 0.9901 - val_loss: 0.0272 -
val_accuracy: 0.9958
Epoch 2/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0240 - accuracy: 0.9942 - val_loss: 0.0138 -
val_accuracy: 0.9960
Epoch 3/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0178 - accuracy: 0.9949 - val_loss: 0.0136 -
val_accuracy: 0.9958
Epoch 4/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0164 - accuracy: 0.9948 - val_loss: 0.0083 -
val_accuracy: 0.9978
Epoch 5/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0129 - accuracy: 0.9960 - val_loss: 0.0058 -
val_accuracy: 0.9984
Epoch 6/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0113 - accuracy: 0.9964 - val_loss: 0.0060 -
val_accuracy: 0.9991
Epoch 7/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0108 - accuracy: 0.9967 - val_loss: 0.0045 -
val_accuracy: 0.9991
```

```
Epoch 8/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0116 - accuracy: 0.9965 - val_loss: 0.0050 -
val_accuracy: 0.9989
Epoch 9/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0087 - accuracy: 0.9970 - val_loss: 0.0080 -
val_accuracy: 0.9987
Epoch 10/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0093 - accuracy: 0.9974 - val_loss: 0.0055 -
val_accuracy: 0.9987
Epoch 11/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0088 - accuracy: 0.9972 - val_loss: 0.0036 -
val_accuracy: 0.9991
Epoch 12/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0079 - accuracy: 0.9972 - val_loss: 0.0089 -
val_accuracy: 0.9984
Epoch 13/20
180/180 [==============================] - 3s 16ms/step - loss: 0.0061 - accuracy: 0.9980 - val_loss: 0.0031 -
val_accuracy: 0.9996
Epoch 14/20
180/180 [==============================] - 3s 16ms/step - loss: 0.0077 - accuracy: 0.9976 - val_loss: 0.0044 -
val_accuracy: 0.9993
Epoch 15/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0054 - accuracy: 0.9982 - val_loss: 0.0019 -
val_accuracy: 0.9996
Epoch 16/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0059 - accuracy: 0.9978 - val_loss: 0.0010 -
val_accuracy: 0.9998
Epoch 17/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0070 - accuracy: 0.9979 - val_loss: 0.0017 -
val_accuracy: 0.9998
Epoch 18/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0052 - accuracy: 0.9986 - val_loss: 0.0062 -
val_accuracy: 0.9973
Epoch 19/20
180/180 [==============================] - 3s 17ms/step - loss: 0.0059 - accuracy: 0.9983 - val_loss: 7.7297e-04 -
val_accuracy: 1.0000
Epoch 20/20
```

```
180/180 [==============================] - 3s 17ms/step - loss: 0.0046 - accuracy: 0.9988 - val_loss: 0.0043 -
val_accuracy: 0.9996
```

Plot the Accuracy

```
plot_accuracy(history3)
```



Recommendation:

Based on the result, since **model 3** has at least 99% accuracy,we should use **both title and text** to detect fake news.

Part4: Model Evaluation

In this part,we'll test your model performance on unseen test data.

Acquire Test Dataset

```
test_url = "https://github.com/PhilChodrow/PIC16b/blob/master/datasets/fake_news_test.csv?raw=true"
df_test  = pd.read_csv(test_url)
test_dataset=make_dataset(df_test)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/a10033/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Evaluate the best model

```
model3.evaluate(test_dataset)
```

```
225/225 [==============================] - 1s 6ms/step - loss: 0.0202 - accuracy: 0.9944
```

```
[0.020209552720189095, 0.9943872690200806]
```

Summary:

If we used our model as a fake news detector, it has nearly 99% chance of being correct.

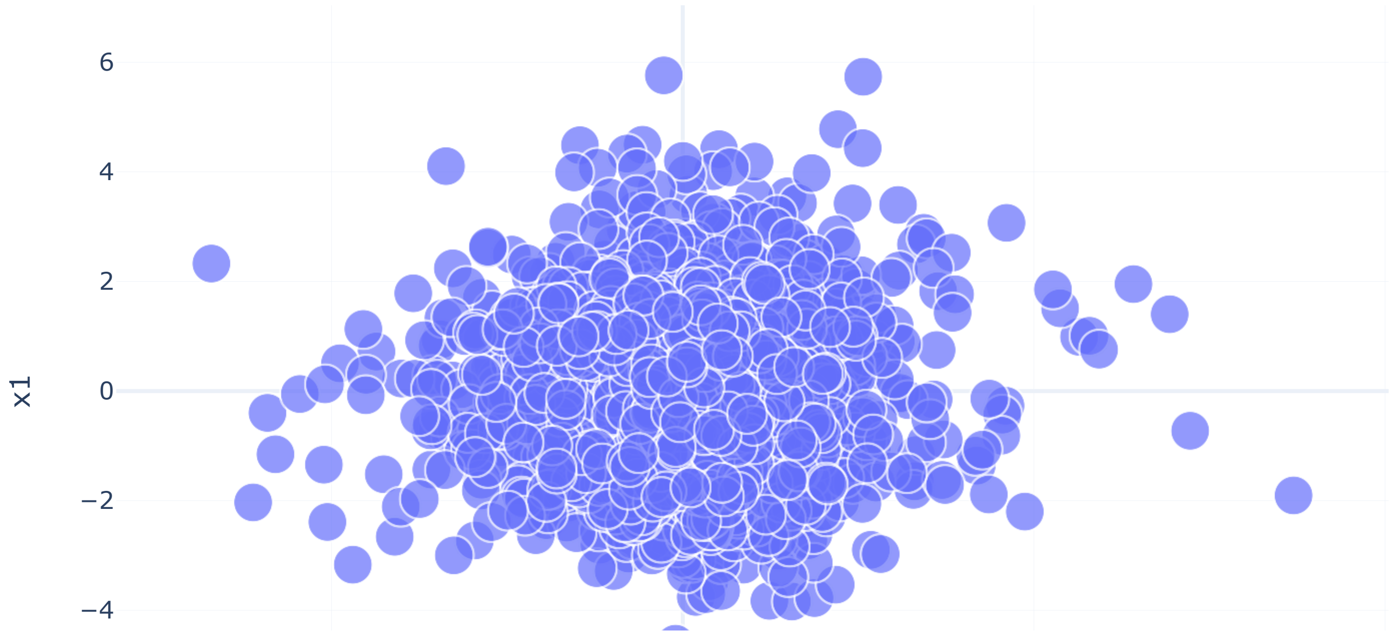
Part5: Embedding Visualization

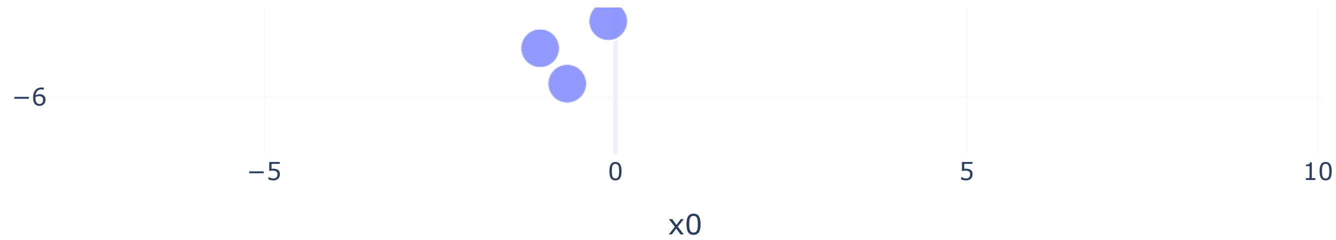
```
weights = model3.get_layer('embedding').get_weights()[0] # get the weights from the embedding layer
vocab = title_vectorize_layer.get_vocabulary()                # get the vocabulary from our data prep for later

pca = PCA(n_components=2)
```

```
weights = pca.fit_transform(weights)

embedding_df = pd.DataFrame({
    'word' : vocab,
    'x0'   : weights[:,0],
    'x1'   : weights[:,1]
})
```

Plot

```
fig = px.scatter(embedding_df,
                 x = "x0",
                 y = "x1",
                 size = [2]*len(embedding_df),
                 hover_name = "word")

fig.show(renderer="notebook")
```

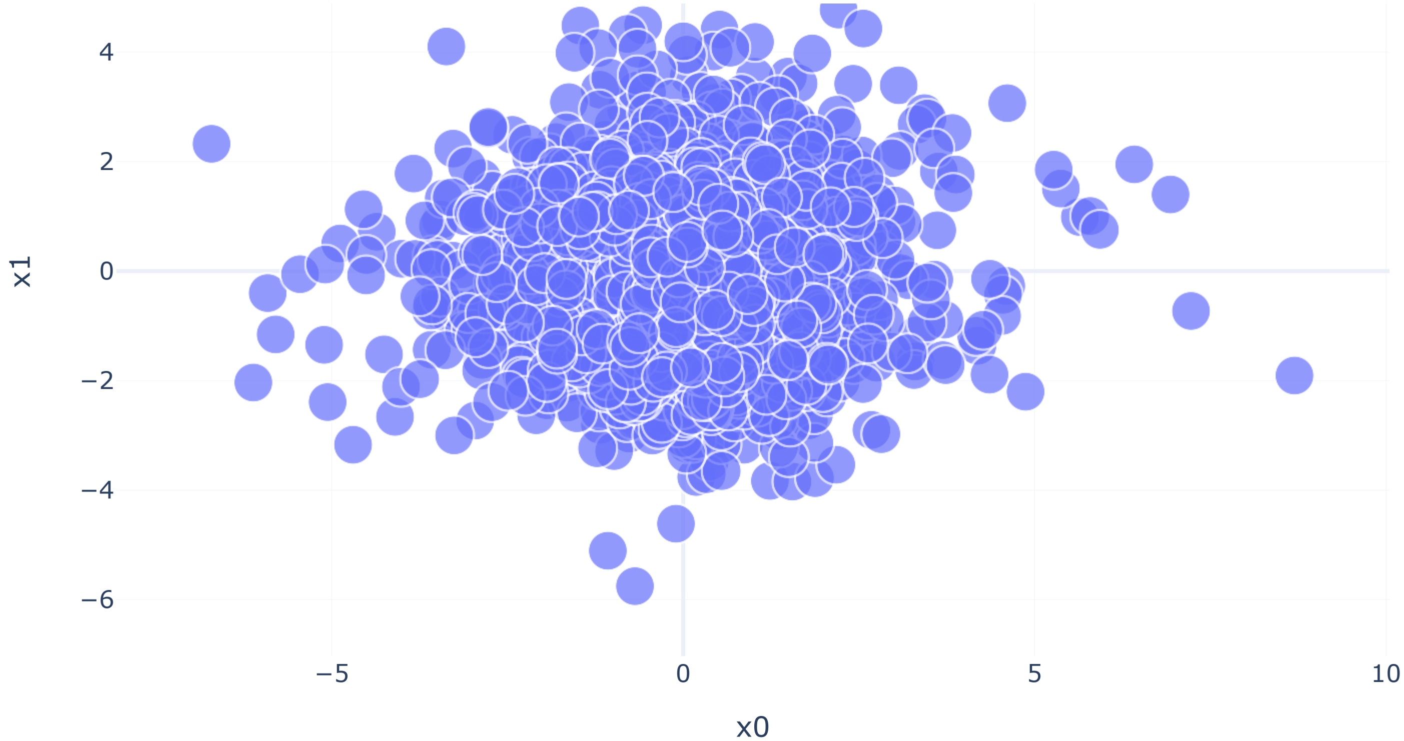





```
weights = model3.get_layer('embedding').get_weights()[0] # get the weights from the embedding layer
vocab = text_vectorize_layer.get_vocabulary()                # get the vocabulary from our data prep for later

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

embedding_df = pd.DataFrame({
    'word' : vocab,
    'x0'   : weights[:,0],
    'x1'   : weights[:,1]
})
```

Plot

```
fig = px.scatter(embedding_df,
                 x = "x0",
                 y = "x1",
                 size = [2]*len(embedding_df),
                 hover_name = "word")

fig.show(renderer="notebook")
```



Observation:

- From the visualized graph, We can find that **campagigns,local and republican** are very close to each other. We can think that political or social news associated with them is fake news
- Also, we ca find that **Steve** and **Tweet** are so close, that may means that combing these two words are probaly fake news.