Text Classification

For this assignment, I am using a data set taken from https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset.

This data set contained two csv files: one containing a list of real news articles, and the other one containing a list of fake ones. I joined those two files into one, and also added a column with the corresponding label. Lastly, I deleted half of the data to make the data set smaller. Now, this data set contains 10,000 news articles, with half of them having the label 'real' and the other half having the label 'fake'.

The purpose of the models I will create is to accurately predict if a news article is real or fake.

Import Required Package

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
import numpy as np
import pandas as pd
```

Read Data

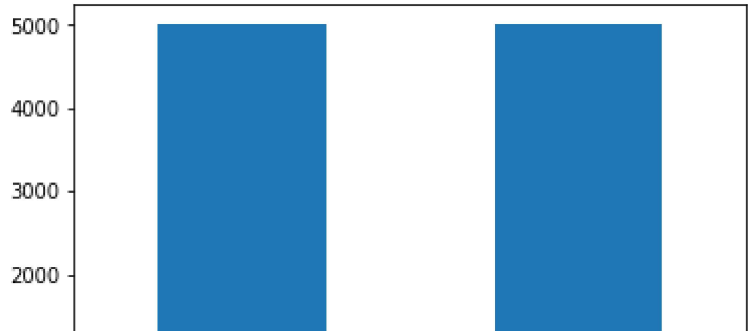
```
df = pd.read_csv("News_Data.csv", header=0, usecols=[0,1], encoding='latin-1')
```

Class Distributions

Looking at the graph below, we can notice that both the target classes have an equal number of instances in the data set.

```
df['news_type'].value_counts().plot(kind="bar")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2aa6e19b10>
```



Divide into Train and Test

```
# split df into train and test
i = np.random.rand(len(df)) < 0.8
train_data = df.text[i]
train_labels = df.news_type[i]
test_data = df.text[~i]
test_labels = df.news_type[~i]
```

Fit the tokenizer on the training data

```
tokenizer = Tokenizer(num_words=2000)
tokenizer.fit_on_texts(train_data)

x_train = tokenizer.texts_to_matrix(train_data, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test_data, mode='tfidf')

encoder = LabelEncoder()
y_train = encoder.fit_transform(train_labels)
y_test = encoder.transform(test_labels)

print("train shapes:", x_train.shape, y_train.shape)
print("test shapes:", x_test.shape, y_test.shape)
```

```
train shapes: (7972, 2000) (7972,)
test shapes: (2028, 2000) (2028,)
```

Build a Sequential Model

```
seq_model = models.Sequential()
seq_model.add(layers.Dense(16, input_dim=2000, kernel_initializer='normal', activation='relu'
seq_model.add(layers.Dense(16, input_dim=2000, kernel_initializer='normal', activation='relu'
seq_model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))
```

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

history = seq_model.fit(x_train, y_train, batch_size=128, epochs=20, verbose=1, validation_sp
```

```
Epoch 1/20
57/57 [==============================] - 1s 8ms/step - loss: 0.2620 - accuracy: 0.9614 -
Epoch 2/20
57/57 [==============================] - 0s 6ms/step - loss: 0.0196 - accuracy: 0.9974 -
Epoch 3/20
57/57 [==============================] - 0s 5ms/step - loss: 0.0029 - accuracy: 0.9997 -
Epoch 4/20
57/57 [==============================] - 0s 5ms/step - loss: 4.6075e-04 - accuracy: 1.00
Epoch 5/20
57/57 [==============================] - 0s 6ms/step - loss: 8.4941e-05 - accuracy: 1.00
Epoch 6/20
57/57 [==============================] - 0s 5ms/step - loss: 1.6432e-05 - accuracy: 1.00
Epoch 7/20
57/57 [==============================] - 0s 5ms/step - loss: 4.8256e-06 - accuracy: 1.00
Epoch 8/20
57/57 [==============================] - 0s 5ms/step - loss: 2.0331e-06 - accuracy: 1.00
Epoch 9/20
57/57 [==============================] - 0s 5ms/step - loss: 8.3403e-07 - accuracy: 1.00
Epoch 10/20
57/57 [==============================] - 0s 5ms/step - loss: 4.4494e-07 - accuracy: 1.00
Epoch 11/20
57/57 [==============================] - 0s 5ms/step - loss: 2.5754e-07 - accuracy: 1.00
Epoch 12/20
57/57 [==============================] - 0s 6ms/step - loss: 1.4809e-07 - accuracy: 1.00
Epoch 13/20
57/57 [==============================] - 0s 5ms/step - loss: 8.5642e-08 - accuracy: 1.00
Epoch 14/20
57/57 [==============================] - 0s 5ms/step - loss: 5.1168e-08 - accuracy: 1.00
Epoch 15/20
57/57 [==============================] - 0s 6ms/step - loss: 2.8502e-08 - accuracy: 1.00
Epoch 16/20
57/57 [==============================] - 0s 5ms/step - loss: 1.7714e-08 - accuracy: 1.00
Epoch 17/20
57/57 [==============================] - 0s 5ms/step - loss: 1.2621e-08 - accuracy: 1.00
Epoch 18/20
57/57 [==============================] - 0s 5ms/step - loss: 8.2340e-09 - accuracy: 1.00
Epoch 19/20
57/57 [==============================] - 0s 5ms/step - loss: 5.7326e-09 - accuracy: 1.00
Epoch 20/20
57/57 [==============================] - 0s 5ms/step - loss: 4.1784e-09 - accuracy: 1.00
```

Predict on the test data

```
pred1 = seq_model.predict(x_test)
pred_labels1 = [1 if p>0.5 else 0 for p in pred1]
```

```
64/64 [==============================] - 0s 2ms/step
```

Evaluate

```
print(classification_report(y_test, pred_labels1))
```

```
                  precision    recall  f1-score   support

               0       1.00      0.95      0.98      1038
               1       0.95      1.00      0.98       990

        accuracy                           0.98      2028
       macro avg       0.98      0.98      0.98      2028
    weighted avg       0.98      0.98      0.98      2028
```

Build a CNN model

```
cnn_model1 = models.Sequential()
cnn_model1.add(layers.Embedding(5000, 128, input_length=2000))
cnn_model1.add(layers.Conv1D(32, 7, activation='relu'))
cnn_model1.add(layers.MaxPooling1D(5))
cnn_model1.add(layers.GlobalMaxPooling1D())
cnn_model1.add(layers.Dense(1))

cnn_model1.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

history = cnn_model1.fit(x_train, y_train, batch_size=128, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
50/50 [==============================] - 123s 2s/step - loss: 0.7939 - accuracy: 0.5739
Epoch 2/10
50/50 [==============================] - 97s 2s/step - loss: 0.5821 - accuracy: 0.6545 -
Epoch 3/10
50/50 [==============================] - 95s 2s/step - loss: 0.5344 - accuracy: 0.7022 -
Epoch 4/10
50/50 [==============================] - 98s 2s/step - loss: 0.5251 - accuracy: 0.7135 -
Epoch 5/10
50/50 [==============================] - 97s 2s/step - loss: 0.5126 - accuracy: 0.7370 -
Epoch 6/10
50/50 [==============================] - 94s 2s/step - loss: 0.4956 - accuracy: 0.7541 -
Epoch 7/10
50/50 [==============================] - 93s 2s/step - loss: 0.4976 - accuracy: 0.7682 -
Epoch 8/10
50/50 [==============================] - 92s 2s/step - loss: 0.4739 - accuracy: 0.7707 -
Epoch 9/10
```

```
50/50 [==============================] - 92s 2s/step - loss: 0.4806 - accuracy: 0.7731 -
Epoch 10/10
50/50 [==============================] - 93s 2s/step - loss: 0.4668 - accuracy: 0.7823 -
```

Predict on the test data

```
pred2 = cnn_model1.predict(x_test)
pred_labels2 = [1 if p>0.5 else 0 for p in pred2]
```

```
64/64 [==============================] - 6s 96ms/step
```

Evaluate

```
print(classification_report(y_test, pred_labels2))
```

```
                  precision    recall  f1-score   support

               0       0.83      0.53      0.65      1038
               1       0.64      0.88      0.74       990

        accuracy                           0.70      2028
       macro avg       0.74      0.71      0.70      2028
    weighted avg       0.74      0.70      0.70      2028
```

Build a second CNN model with a different embedding

```
cnn_model2 = models.Sequential()
cnn_model2.add(layers.Embedding(5000, 128, input_length=2000))
cnn_model2.add(layers.Conv1D(32, 7, activation='relu'))
cnn_model2.add(layers.MaxPooling1D(5))
cnn_model2.add(layers.Conv1D(32, 7, activation='relu'))
cnn_model2.add(layers.GlobalMaxPooling1D())
cnn_model2.add(layers.Dense(1))

cnn_model2.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

history = cnn_model2.fit(x_train, y_train, batch_size=128, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
50/50 [==============================] - 98s 2s/step - loss: 0.6949 - accuracy: 0.6070 -
Epoch 2/10
50/50 [==============================] - 95s 2s/step - loss: 0.5384 - accuracy: 0.7055 -
```

```
Epoch 3/10
50/50 [==============================] - 95s 2s/step - loss: 0.3273 - accuracy: 0.8793 -
Epoch 4/10
50/50 [==============================] - 94s 2s/step - loss: 0.2103 - accuracy: 0.9343 -
Epoch 5/10
50/50 [==============================] - 94s 2s/step - loss: 0.1387 - accuracy: 0.9619 -
Epoch 6/10
50/50 [==============================] - 93s 2s/step - loss: 0.1398 - accuracy: 0.9603 -
Epoch 7/10
50/50 [==============================] - 93s 2s/step - loss: 0.1100 - accuracy: 0.9708 -
Epoch 8/10
50/50 [==============================] - 93s 2s/step - loss: 0.0978 - accuracy: 0.9730 -
Epoch 9/10
50/50 [==============================] - 92s 2s/step - loss: 0.0990 - accuracy: 0.9755 -
Epoch 10/10
50/50 [==============================] - 92s 2s/step - loss: 0.0940 - accuracy: 0.9807 -
```

Predict on the test data

```
pred3 = cnn_model2.predict(x_test)
pred_labels3 = [1 if p>0.5 else 0 for p in pred3]
```

```
64/64 [==============================] - 7s 103ms/step
```

Evaluate

```
print(classification_report(y_test, pred_labels3))
```

```
                  precision    recall  f1-score   support

               0       0.99      0.95      0.97      1038
               1       0.95      0.99      0.97       990

        accuracy                           0.97      2028
       macro avg       0.97      0.97      0.97      2028
    weighted avg       0.97      0.97      0.97      2028
```

Build a third CNN model with a different embedding

```
cnn_model3 = models.Sequential()
cnn_model3.add(layers.Embedding(5000, 256, input_length=2000))
cnn_model3.add(layers.Conv1D(32, 7, activation='relu'))
cnn_model3.add(layers.MaxPooling1D(5))
```

```
cnn_model3.add(layers.Conv1D(32, 7, activation='relu'))
cnn_model3.add(layers.GlobalMaxPooling1D())
cnn_model3.add(layers.Dense(1))

cnn_model3.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

history = cnn_model3.fit(x_train, y_train, batch_size=128, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
50/50 [==============================] - 181s 4s/step - loss: 0.6853 - accuracy: 0.6028
Epoch 2/10
50/50 [==============================] - 189s 4s/step - loss: 0.5097 - accuracy: 0.7613
Epoch 3/10
50/50 [==============================] - 196s 4s/step - loss: 0.2881 - accuracy: 0.9163
Epoch 4/10
50/50 [==============================] - 179s 4s/step - loss: 0.1614 - accuracy: 0.9577
Epoch 5/10
50/50 [==============================] - 178s 4s/step - loss: 0.1822 - accuracy: 0.9520
Epoch 6/10
50/50 [==============================] - 176s 4s/step - loss: 0.1298 - accuracy: 0.9690
Epoch 7/10
50/50 [==============================] - 177s 4s/step - loss: 0.1060 - accuracy: 0.9732
Epoch 8/10
50/50 [==============================] - 185s 4s/step - loss: 0.1251 - accuracy: 0.9743
Epoch 9/10
50/50 [==============================] - 175s 4s/step - loss: 0.1126 - accuracy: 0.9771
Epoch 10/10
50/50 [==============================] - 177s 4s/step - loss: 0.0833 - accuracy: 0.9802
```

Predict on the test data

```
pred4 = cnn_model3.predict(x_test)
pred_labels4 = [1 if p>0.5 else 0 for p in pred4]
```

```
64/64 [==============================] - 13s 203ms/step
```

Evaluate

```
print(classification_report(y_test, pred_labels4))
```

```
              precision    recall  f1-score   support

           0       1.00      0.93      0.96      1038
           1       0.93      1.00      0.96       990

    accuracy                           0.96      2028
```

```
   macro avg       0.96      0.96      0.96      2028
weighted avg       0.96      0.96      0.96      2028
```

Build a fourth CNN model with a different embedding

```
cnn_model4 = models.Sequential()
cnn_model4.add(layers.Embedding(5000, 256, input_length=2000))
cnn_model4.add(layers.Conv1D(32, 7, activation='relu'))
cnn_model4.add(layers.MaxPooling1D(5))
cnn_model4.add(layers.Conv1D(32, 7, activation='relu'))
cnn_model4.add(layers.MaxPooling1D(5))
cnn_model4.add(layers.Conv1D(32, 7, activation='relu'))
cnn_model4.add(layers.GlobalMaxPooling1D())
cnn_model4.add(layers.Dense(1))

cnn_model4.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

history = cnn_model4.fit(x_train, y_train, batch_size=128, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
50/50 [==============================] - 183s 4s/step - loss: 0.7224 - accuracy: 0.6026
Epoch 2/10
50/50 [==============================] - 181s 4s/step - loss: 1.1658 - accuracy: 0.6602
Epoch 3/10
50/50 [==============================] - 182s 4s/step - loss: 0.4868 - accuracy: 0.8346
Epoch 4/10
50/50 [==============================] - 183s 4s/step - loss: 0.3759 - accuracy: 0.9131
Epoch 5/10
50/50 [==============================] - 181s 4s/step - loss: 0.1930 - accuracy: 0.9415
Epoch 6/10
50/50 [==============================] - 184s 4s/step - loss: 0.2632 - accuracy: 0.9479
Epoch 7/10
50/50 [==============================] - 190s 4s/step - loss: 0.1692 - accuracy: 0.9635
Epoch 8/10
50/50 [==============================] - 182s 4s/step - loss: 0.1491 - accuracy: 0.9606
Epoch 9/10
50/50 [==============================] - 183s 4s/step - loss: 0.0923 - accuracy: 0.9787
Epoch 10/10
50/50 [==============================] - 187s 4s/step - loss: 0.0784 - accuracy: 0.9823
```

Predict on the test data

```
pred5 = cnn_model4.predict(x_test)
pred_labels5 = [1 if p>0.5 else 0 for p in pred5]
```

```
64/64 [==============================] - 13s 196ms/step
```

Evaluate

```
print(classification_report(y_test, pred_labels5))
```

```
                  precision    recall  f1-score   support

               0       0.80      0.98      0.88      1038
               1       0.97      0.75      0.85       990

        accuracy                           0.87      2028
       macro avg       0.89      0.86      0.86      2028
    weighted avg       0.89      0.87      0.87      2028
```

Analysis of the performance of different approaches

First, I created a sequential model with three dense layers. In that model, Layer 1 and 2 had 16 nodes each, with a 'relu' activation, and Layer 3 only had 1 node, with a 'sigmoid' activation. This model gave an overall accuracy of 98 percent.

Then, I decided to build some CNN models with different embedding approaches.

First CNN model I made contained 1 embedding layer that output a 3D floating-point tensor with an embedding dimension of 128. It also contained a Conv1D layer with 32 filters and a kernel size of 7x7. Lastly, it had a MaxPooling layer with a pool size of 5, and a dense layer with 1 node that was used to classify the target classes. This model only gave an accuracy of 70 percent.

For the second CNN model, I kept most of the arguments same. I just added another Conv1D layer with 32 filters and a kernel size of 7x7. This model gave an accuracy of 97 percent, which is an increase of 27 percent from the previous model.

For the third CNN model, I decided to keep the same number of layers, but change the embedding dimension fromm 128 to 256. Instead of increasing the accuracy, it actually decreased it by 1 percent, giving an accuracy of 96 percent.

Lastly, for the fouth CNN model, I decided to add another Convc1D layer because I thought if adding a Conv1D layer increased the accuracy from first CNN model to second, maybe it will increase the accuracy from the third model to fourth. But, it actually did the opposite. It brought the accuracy down to 87.

Colab paid products - Cancel contracts here

0s completed at 3:51 PM