```
!pip install kaggle
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: kaggle in /usr/local/lib/python3.9/dist-packages (1.5.13)
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from kaggle) (2.27.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.9/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.9/dist-packages (from kaggle) (8.0.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.9/dist-packages (from kaggle) (1.26.15)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.9/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from kaggle) (4.65.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.9/dist-packages (from kaggle) (2022.12.7)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.9/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests->kaggle) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests->kaggle) (3.4)
```

```
from google.colab import files
files.upload()
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 65 bytes, last modified: 3/28/2023 - 100% done

```
Saving kaggle.json to kaggle.json
{'kaggle.json':
b'{"username":"zakenmaru","key":"7757c32ff01f25a734cbaec265cafafe"}'}
```

```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
mkdir: cannot create directory '/root/.kaggle': File exists
```

```
!chmod 600 ~/.kaggle/kaggle.json
```

Saved successfully!

```
abhshahane/fake-news-classification
```

```
Downloading fake-news-classification.zip to /content
 98% 90.0M/92.1M [00:03<00:00, 27.5MB/s]
100% 92.1M/92.1M [00:03<00:00, 28.0MB/s]
```

```
!unzip fake-news-classification.zip
```

```
Archive:  fake-news-classification.zip
  inflating: WELFake_Dataset.csv
```

```
import pandas as pd

df = pd.read_csv('WELFake_Dataset.csv')
```

```
df
```

	Unnamed: 0	title	text	label
0	0	LAW ENFORCEMENT ON HIGH ALERT Following Threat...	No comment is expected from Barack Obama Membe...	1
1	1	NaN	Did they post their votes for Hillary already?	1
2	2	UNBELIEVABLE! OBAMA'S ATTORNEY GENERAL SAYS MO...	Now, most of the demonstrators gathered last ...	1
3	3	Bobby Jindal, raised Hindu, uses story of Chri...	A dozen politically active pastors came here f...	0
4	4	SATAN 2: Russia unvelis an image of its terrif...	The RS-28 Sarmat missile, dubbed Satan 2, will...	1
...	...	...	...	...
72129	72129	Russians steal research on Trump in hack of U....	WASHINGTON (Reuters) - Hackers believed to be ...	0
72130	72130	WATCH: Giuliani Demands That	You know, because in	1

```
# get cols of the df
df.columns
```

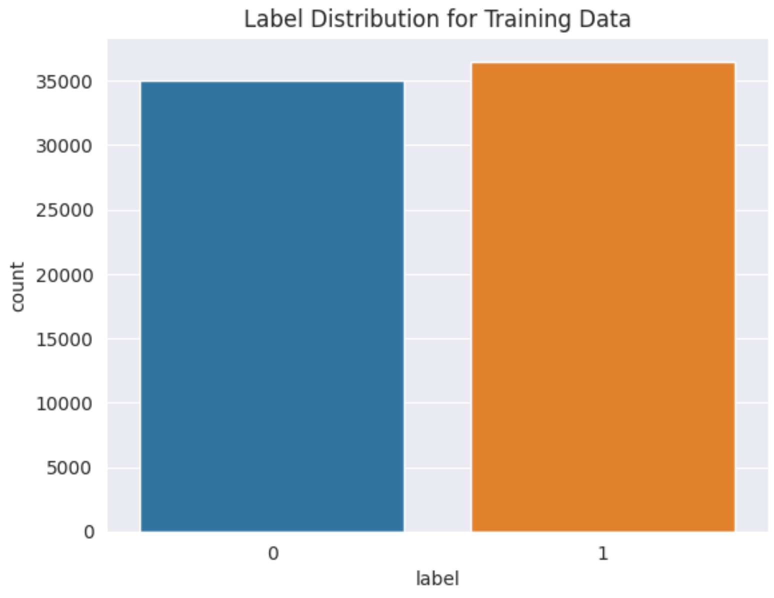
```
# sentiment is the y value, review is the x value. We split with an 80/20 split
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import numpy as np
# Filter out any rows with NaN values
df = df.dropna()

# Separate features and target variable
X = df[['title', 'text']]
y = df['label']

# Convert categorical variables using Label Encoding
label_encoder = LabelEncoder()
X['title'] = label_encoder.fit_transform(X['title'])
X['text'] = label_encoder.fit_transform(X['text'])
y = label_encoder.fit_transform(y)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# make sure that all text is string
X_train = X_train.astype(str)
X_test = X_test.astype(str)
y_train = np.asarray(y_train).astype('float32').reshape((-1,1))
y_test = np.asarray(y_test).astype('float32').reshape((-1,1))
```

```
<ipython-input-13-531550a5e943>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

                           ation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
                           t_transform(X['title'])
                           :15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  X['text'] = label_encoder.fit_transform(X['text'])
```

Saved successfully!

```
import seaborn as sns

sns.set_style('darkgrid')
sns.countplot(x='label', data=df).set(title='Label Distribution for Training Data')
```

```
[Text(0.5, 1.0, 'Label Distribution for Training Data')]
```



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

This data set contains real news and fake news, with titles and text being assigned to each. It contains 4 columns, but as the first is just an identifier, we only consider the other 3. These are: Title (about the text news heading); Text (about the news content); and Label (0 = fake and 1 = real).

The model should be able to accurately predict whether a given entry is real or fake news given the title and text of the entry. There is an 80/20 split between training/testing data for the model to predict on.

```
# Sequential

from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Embedding, Flatten
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from tensorflow.keras.preprocessing.sequence import pad_sequences
import pandas as pd

# Tokenize and pad sequences
max_words = 10000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_train['title'] + ' ' + X_train['text'])
X_train_seq = tokenizer.texts_to_sequences(X_train['title'] + ' ' + X_train['text'])
X_test_seq = tokenizer.texts_to_sequences(X_test['title'] + ' ' + X_test['text'])
max_seq_length = max(len(x) for x in X_train_seq)
X_train_padded = pad_sequences(X_train_seq, maxlen=max_seq_length)
X_test_padded = pad_sequences(X_test_seq, maxlen=max_seq_length)

# Define model
model = Sequential()
model.add(Flatten())
Saved successfully!                  relu'))
                                     elu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

# Train model
model.fit(X_train_padded, y_train, epochs=15, batch_size=16, validation_data=(X_test_padded, y_test))
```

```
Epoch 1/15
3577/3577 [==============================] - 32s 8ms/step - loss: 1.8217 - accuracy: 0.6226 - val_loss: 0.7168 - val_accuracy: 0.5239
Epoch 2/15
3577/3577 [==============================] - 16s 4ms/step - loss: 0.6434 - accuracy: 0.6485 - val_loss: 0.6881 - val_accuracy: 0.5553
Epoch 3/15
3577/3577 [==============================] - 17s 5ms/step - loss: 0.6503 - accuracy: 0.6500 - val_loss: 0.6974 - val_accuracy: 0.5553
Epoch 4/15
3577/3577 [==============================] - 17s 5ms/step - loss: 0.6418 - accuracy: 0.6500 - val_loss: 0.6761 - val_accuracy: 0.5553
Epoch 5/15
3577/3577 [==============================] - 16s 4ms/step - loss: 0.6416 - accuracy: 0.6500 - val_loss: 0.6939 - val_accuracy: 0.5553
Epoch 6/15
3577/3577 [==============================] - 18s 5ms/step - loss: 0.6415 - accuracy: 0.6498 - val_loss: 0.6778 - val_accuracy: 0.5553
Epoch 7/15
3577/3577 [==============================] - 17s 5ms/step - loss: 0.6404 - accuracy: 0.6500 - val_loss: 0.6831 - val_accuracy: 0.5553
Epoch 8/15
3577/3577 [==============================] - 16s 4ms/step - loss: 0.6406 - accuracy: 0.6500 - val_loss: 0.6826 - val_accuracy: 0.5553
Epoch 9/15
3577/3577 [==============================] - 17s 5ms/step - loss: 0.6402 - accuracy: 0.6500 - val_loss: 0.6782 - val_accuracy: 0.5553
Epoch 10/15
3577/3577 [==============================] - 16s 4ms/step - loss: 0.6418 - accuracy: 0.6500 - val_loss: 0.6849 - val_accuracy: 0.5553
Epoch 11/15
3577/3577 [==============================] - 16s 4ms/step - loss: 0.6402 - accuracy: 0.6500 - val_loss: 0.6825 - val_accuracy: 0.5553
Epoch 12/15
3577/3577 [==============================] - 17s 5ms/step - loss: 0.6402 - accuracy: 0.6500 - val_loss: 0.6843 - val_accuracy: 0.5553
Epoch 13/15
3577/3577 [==============================] - 16s 4ms/step - loss: 0.6420 - accuracy: 0.6500 - val_loss: 0.6799 - val_accuracy: 0.5553
Epoch 14/15
3577/3577 [==============================] - 17s 5ms/step - loss: 0.6401 - accuracy: 0.6500 - val_loss: 0.6862 - val_accuracy: 0.5553
Epoch 15/15
3577/3577 [==============================] - 17s 5ms/step - loss: 0.6401 - accuracy: 0.6500 - val_loss: 0.6829 - val_accuracy: 0.5553
<keras.callbacks.History at 0x7f1e7bda5eb0>
```

```
# CNN

from keras.layers import Dense, Embedding, Conv1D, MaxPooling1D, GlobalMaxPooling1D

```

```
# Tokenize and pad sequences; higher max word count and seq length
max_words = 10000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_train['title'] + ' ' + X_train['text'])
X_train_seq = tokenizer.texts_to_sequences(X_train['title'] + ' ' + X_train['text'])
X_test_seq = tokenizer.texts_to_sequences(X_test['title'] + ' ' + X_test['text'])
max_seq_length = 1000  # Increase the max sequence length
X_train_padded = pad_sequences(X_train_seq, maxlen=max_seq_length)
X_test_padded = pad_sequences(X_test_seq, maxlen=max_seq_length)

# Define model
model = Sequential()
model.add(Embedding(max_words, output_dim=50, input_length=max_seq_length, mask_zero=True))
model.add(Conv1D(32, 5, activation='relu'))
model.add(MaxPooling1D(5))
model.add(Conv1D(64, 5, activation='relu'))
model.add(MaxPooling1D(5))
model.add(Conv1D(128, 5, activation='relu'))
model.add(MaxPooling1D(5))
model.add(Conv1D(256, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(1, activation='sigmoid'))

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

# Train model
model.fit(X_train_padded, y_train, epochs=15, batch_size=16, validation_data=(X_test_padded, y_test))
```

```
Epoch 1/15
3577/3577 [==============================] - 32s 8ms/step - loss: 0.6273 - accuracy: 0.6048 - val_loss: 0.6899 - val_accuracy: 0.5082
Saved successfully!            ==========] - 30s 8ms/step - loss: 0.5879 - accuracy: 0.6291 - val_loss: 0.6933 - val_accuracy: 0.5083

3577/3577 [==============================] - 29s 8ms/step - loss: 0.5881 - accuracy: 0.6292 - val_loss: 0.6896 - val_accuracy: 0.5083
Epoch 4/15
3577/3577 [==============================] - 32s 9ms/step - loss: 0.5876 - accuracy: 0.6293 - val_loss: 0.6923 - val_accuracy: 0.5083
Epoch 5/15
3577/3577 [==============================] - 27s 7ms/step - loss: 0.5878 - accuracy: 0.6293 - val_loss: 0.6895 - val_accuracy: 0.5083
Epoch 6/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5876 - accuracy: 0.6293 - val_loss: 0.6958 - val_accuracy: 0.5083
Epoch 7/15
3577/3577 [==============================] - 27s 7ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6903 - val_accuracy: 0.5083
Epoch 8/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6929 - val_accuracy: 0.5083
Epoch 9/15
3577/3577 [==============================] - 27s 7ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6931 - val_accuracy: 0.5083
Epoch 10/15
3577/3577 [==============================] - 30s 8ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6908 - val_accuracy: 0.5083
Epoch 11/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6932 - val_accuracy: 0.5083
Epoch 12/15
3577/3577 [==============================] - 27s 7ms/step - loss: 0.5877 - accuracy: 0.6293 - val_loss: 0.6928 - val_accuracy: 0.5083
Epoch 13/15
3577/3577 [==============================] - 27s 7ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6936 - val_accuracy: 0.5083
Epoch 14/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6926 - val_accuracy: 0.5083
Epoch 15/15
3577/3577 [==============================] - 28s 8ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6917 - val_accuracy: 0.5083
<keras.callbacks.History at 0x7f1dcaeeedf0>
```

```
# Embeddings

from gensim.models import Word2Vec

# Concatenate title and text columns
X_train_concat = X_train['title'] + ' ' + X_train['text']
X_test_concat = X_test['title'] + ' ' + X_test['text']

# Tokenize sequences
max_words = 10000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_train_concat)
X_train_seq = tokenizer.texts_to_sequences(X_train_concat)
X_test_seq = tokenizer.texts_to_sequences(X_test_concat)

# Pad sequences
max_seq_length = 1000
X_train_padded = pad_sequences(X_train_seq, maxlen=max_seq_length)
```

```
X_test_padded = pad_sequences(X_test_seq, maxlen=max_seq_length)

# Train Word2Vec model
embedding_dim = 50
w2v_model = Word2Vec(sentences=X_train_seq, window=5, min_count=1, workers=4)

# Initialize embedding matrix with zeros
embedding_matrix = np.zeros((max_words, embedding_dim))

# Build embedding matrix
for word, i in tokenizer.word_index.items():
    if i < max_words:
        if word in w2v_model.wv.key_to_index:
            embedding_matrix[i] = w2v_model.wv[word]


# Define model
model = Sequential()
model.add(Embedding(max_words, output_dim=50, input_length=max_seq_length, mask_zero=True))
model.add(Conv1D(32, 5, activation='relu'))
model.add(MaxPooling1D(5))
model.add(Conv1D(64, 5, activation='relu'))
model.add(MaxPooling1D(5))
model.add(Conv1D(128, 5, activation='relu'))
model.add(MaxPooling1D(5))
model.add(Conv1D(256, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(1, activation='sigmoid'))


# Compile model
Saved successfully!            oss='binary_crossentropy', metrics=['accuracy'])

# Train model
model.fit(X_train_padded, y_train, epochs=15, batch_size=16, validation_data=(X_test_padded, y_test))
```

```
Epoch 1/15
3577/3577 [==============================] - 33s 9ms/step - loss: 0.6377 - accuracy: 0.5902 - val_loss: 0.6875 - val_accuracy: 0.5083
Epoch 2/15
3577/3577 [==============================] - 28s 8ms/step - loss: 0.5901 - accuracy: 0.6272 - val_loss: 0.6941 - val_accuracy: 0.5083
Epoch 3/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5881 - accuracy: 0.6292 - val_loss: 0.6921 - val_accuracy: 0.5084
Epoch 4/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5875 - accuracy: 0.6293 - val_loss: 0.6936 - val_accuracy: 0.5083
Epoch 5/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5875 - accuracy: 0.6293 - val_loss: 0.6913 - val_accuracy: 0.5083
Epoch 6/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5875 - accuracy: 0.6293 - val_loss: 0.6924 - val_accuracy: 0.5083
Epoch 7/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6917 - val_accuracy: 0.5083
Epoch 8/15
3577/3577 [==============================] - 28s 8ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6956 - val_accuracy: 0.5083
Epoch 9/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5875 - accuracy: 0.6293 - val_loss: 0.6946 - val_accuracy: 0.5083
Epoch 10/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6902 - val_accuracy: 0.5083
Epoch 11/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6921 - val_accuracy: 0.5083
Epoch 12/15
3577/3577 [==============================] - 27s 7ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6945 - val_accuracy: 0.5083
Epoch 13/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6909 - val_accuracy: 0.5083
Epoch 14/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6911 - val_accuracy: 0.5083
Epoch 15/15
3577/3577 [==============================] - 27s 8ms/step - loss: 0.5874 - accuracy: 0.6293 - val_loss: 0.6927 - val_accuracy: 0.5083
<keras.callbacks.History at 0x7f1df255ba00>
```

Analysis

I tried three different approaches when it came to classification of real/fake news - Sequential, CNN, and embedding.

For all 3, they used 15 epochs and 16 batch size.

Sequential

Sequential was the simplest model, but had the highest accuracy out of the three. The end accuracy was 0.65, and the highest of the lot. There were no embedding layers, only Dense.

CNN

I opted for a CNN over an RNN as this data isn't sequential/time series, thus an RNN isn't proper use case for this. This yielded an accuracy of 0.6293, and used the MaxPooling functions for the purposes of the network.

Embedding

The Embedded CNN used the Word2vec embedding to learn word associations. After training the w2v model, the weights of this model were used when applying it to the actual CNN. This also yielded an accuracy of 0.6293, tieing it with CNN and below Sequential.

Conclusion

Overall, I had a good time working with the different models to try and figure out what was going on with each during this process. However, I was surprised at the Sequential performance beating out the other models.

Saved successfully! ×

✓ 7m 26s completed at 4:33 PM ● ×