Movie Genre Text Classification

This notebook uses a Kaggle dataset containing pieces of about 22,500 scripts. These script chunks will be used to predict the main movie genre for the given script.

Data set can be found at : https://www.kaggle.com/datasets/lykin22/movie-genre-data?select=kaggle_movie_train.csv

Code inspired by Karen Mazidi's GitHub notebooks:

https://github.com/kjmazidi/NLP/blob/master/Part_6-Deep%20Learning/Chapter_23_Keras/Keras_imbd_1_Dense_Sequential.ipynb

https://github.com/kjmazidi/NLP/blob/master/Part_6-Deep%20Learning/Chapter_24_DL_variations/Keras_imdb_2_RNN.ipynb

https://github.com/kjmazidi/NLP/blob/master/Part_6-Deep%20Learning/Chapter_25_Embeddings/Embedding%20Layer.ipynb

Import Data

```
from google.colab import drive
import pandas as pd

```

```
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
cols = ['text','genre']
movies = pd.read_csv('/content/drive/MyDrive/SCHOOL/Human Language Technology/HW/text_classification/kaggle_movie_train.csv',  header=0, usecols=cols, encoding='latin-1')
movies = movies[pd.notnull(movies['text'])]
movies.head()
```

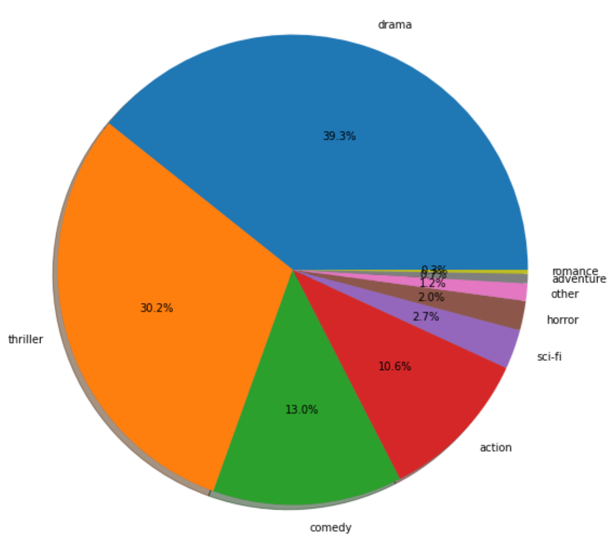
	text	genre
0	eady dead, maybe even wishing he was. INT. 2ND...	thriller
1	t, summa cum laude and all. And I'm about to l...	comedy
2	up Come, I have a surprise.... She takes him ...	drama
3	ded by the two detectives. INT. JEFF'S APARTME...	thriller
4	nd dismounts, just as the other children reach...	drama

Display Target Distribution

```
import matplotlib.pyplot as plt
```

```
genre_labels = movies['genre'].value_counts().index.tolist()
```

```
plt.figure(1, figsize=(20,10))
plt.pie(movies['genre'].value_counts(), labels = genre_labels, shadow = True, autopct='%1.1f%%')
plt.show()
```



Text Preprocessing

There are a lot of potential outcomes for this dataset- many of which aren't representative of much of the data. I want to drop all genres that aren't drama or thriller. I feel like the other genres can fit in these categories anyway.

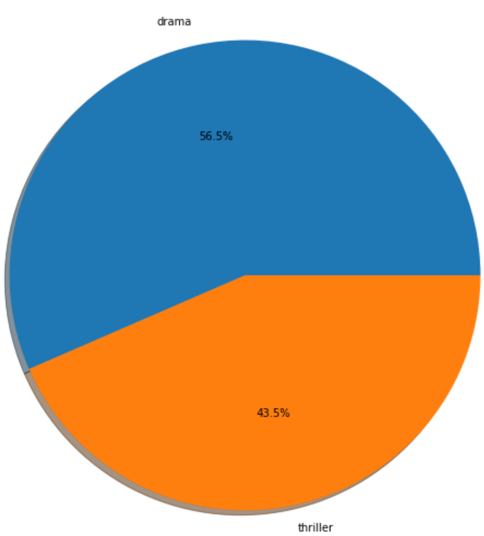
```
movies = movies[movies.genre != 'romance']
movies = movies[movies.genre != 'adventure']
movies = movies[movies.genre != 'other']
movies = movies[movies.genre != 'horror']
movies = movies[movies.genre != 'sci-fi']
movies = movies[movies.genre != 'comedy']
movies = movies[movies.genre != 'action']
```

```
movies
```

	text	genre
0	eady dead, maybe even wishing he was. INT. 2ND...	thriller
2	up Come, I have a surprise.... She takes him ...	drama
3	ded by the two detectives. INT. JEFF'S APARTME...	thriller
4	nd dismounts, just as the other children reach...	drama
5	breadth of the bluff. Gabe pulls out his ancie...	thriller
...	...	...
22571	watching us all. SWISH PAN TO INT. TANGIERS E...	drama
22572	HER and TWO COOKS are standing in a row waitin...	thriller
22574	n in the world to decide what I'm going to do ...	drama
22575	shards. BOJO LAZ! Laz pushes Deke back through...	drama
22576	OTTIE You've got a thing about Ernie's, haven'...	thriller

15697 rows × 2 columns

```
genre_labels = movies['genre'].value_counts().index.tolist()
plt.figure(1, figsize=(20,10))
plt.pie(movies['genre'].value_counts(), labels = genre_labels, shadow = True, autopct='%1.1f%%')
plt.show()
```



The updated data looks much easier to work with since there are no longer 9 potential target values. The model should be able to classify the chunk of movie script into one of the remaining 4 categories: action, drama, thriller, comedy.

```
# convert labels to numeric categorical
# dictionaries to track what is what
movies['genre_id'] = movies['genre'].factorize()[0]
genre_id_df =  movies[['genre', 'genre_id']].drop_duplicates().sort_values('genre_id')
genre_to_id = dict(genre_id_df.values)
id_to_genre = dict(genre_id_df [['genre_id', 'genre']].values)
```

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
#from tensorflow.keras.preprocessing.text import Tokenizer
#from tensorflow.keras import preprocessing
nltk.download('punkt')
stop_words = set(stopwords.words('english'))
```

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

```
def clean_text(x):
    words = [word.lower() for word in word_tokenize(x) if word.isalpha() and not word in stop_words and len(word)<10]
    words = ' '.join(words)
    return words
```

```
for index in movies.index:
    movies['text'][index]= clean_text(movies['text'][index])
```

```
<ipython-input-13-d9c933a3f669>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  movies['text'][index]= clean_text(movies['text'][index])
```

Split Data

```
from sklearn.model_selection import train_test_split
```

```
X = movies['text'].values
y = movies['genre_id'].values
```

```
train_text,test_text, train_labels, test_labels = train_test_split(X, y, test_size=0.2, random_state=0)
```

Sequential Model

The simple sequential model is being used as a baseline.

```
import tensorflow as tf
import numpy as np
from tensorflow.keras import layers, models
from sklearn.feature_extraction.text import CountVectorizer
```

```
# variables
vocab_size = 10000
dimensions = vocab_size
```

```
vectorizer = CountVectorizer()
vectorizer.fit(train_text)

X_train = vectorizer.transform(train_text).toarray()
X_test = vectorizer.transform(test_text).toarray()
```

```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

```
from keras.backend import clear_session
```

```
input_dim = X_train.shape[1]
clear_session()
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(input_dim,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid')) # for binary output
```

```
# compile
model.compile(optimizer='adam', loss='binary_crossentropy',metrics=['accuracy']) # use adam since NOT 1 hot encoded
```

```
# evaluate
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20, batch_size=512)
```

```
Epoch 1/20
25/25 [==============================] - 7s 239ms/step - loss: 0.5661 - accuracy: 0.7077 - val_loss: 0.4007 - val_accuracy: 0.8822
Epoch 2/20
25/25 [==============================] - 5s 209ms/step - loss: 0.2405 - accuracy: 0.9627 - val_loss: 0.2062 - val_accuracy: 0.9443
Epoch 3/20
25/25 [==============================] - 5s 211ms/step - loss: 0.0826 - accuracy: 0.9931 - val_loss: 0.1416 - val_accuracy: 0.9529
Epoch 4/20
25/25 [==============================] - 5s 210ms/step - loss: 0.0333 - accuracy: 0.9988 - val_loss: 0.1245 - val_accuracy: 0.9551
Epoch 5/20
25/25 [==============================] - 5s 212ms/step - loss: 0.0163 - accuracy: 0.9998 - val_loss: 0.1167 - val_accuracy: 0.9580
Epoch 6/20
25/25 [==============================] - 9s 359ms/step - loss: 0.0094 - accuracy: 1.0000 - val_loss: 0.1138 - val_accuracy: 0.9557
Epoch 7/20
25/25 [==============================] - 10s 407ms/step - loss: 0.0061 - accuracy: 1.0000 - val_loss: 0.1130 - val_accuracy: 0.9554
Epoch 8/20
25/25 [==============================] - 9s 378ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.1135 - val_accuracy: 0.9564
Epoch 9/20
25/25 [==============================] - 8s 300ms/step - loss: 0.0031 - accuracy: 1.0000 - val_loss: 0.1141 - val_accuracy: 0.9551
Epoch 10/20
25/25 [==============================] - 5s 208ms/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 0.1143 - val_accuracy: 0.9554
```

```
Epoch 11/20
25/25 [==============================] - 5s 210ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.1146 - val_accuracy: 0.9551
Epoch 12/20
25/25 [==============================] - 5s 209ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.1159 - val_accuracy: 0.9554
Epoch 13/20
25/25 [==============================] - 5s 206ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.1168 - val_accuracy: 0.9557
Epoch 14/20
25/25 [==============================] - 5s 208ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.1174 - val_accuracy: 0.9561
Epoch 15/20
25/25 [==============================] - 5s 205ms/step - loss: 8.7611e-04 - accuracy: 1.0000 - val_loss: 0.1182 - val_accuracy: 0.9554
Epoch 16/20
25/25 [==============================] - 5s 206ms/step - loss: 7.5026e-04 - accuracy: 1.0000 - val_loss: 0.1189 - val_accuracy: 0.9554
Epoch 17/20
25/25 [==============================] - 5s 207ms/step - loss: 6.4928e-04 - accuracy: 1.0000 - val_loss: 0.1197 - val_accuracy: 0.9551
Epoch 18/20
25/25 [==============================] - 5s 210ms/step - loss: 5.6733e-04 - accuracy: 1.0000 - val_loss: 0.1203 - val_accuracy: 0.9548
Epoch 19/20
25/25 [==============================] - 5s 211ms/step - loss: 4.9937e-04 - accuracy: 1.0000 - val_loss: 0.1207 - val_accuracy: 0.9548
Epoch 20/20
25/25 [==============================] - 5s 205ms/step - loss: 4.4281e-04 - accuracy: 1.0000 - val_loss: 0.1219 - val_accuracy: 0.9551
```

```
score = model.evaluate(X_test, y_test, batch_size=512, verbose=1)
print('Accuracy: ', score[1])
```

```
7/7 [==============================] - 1s 77ms/step - loss: 0.1219 - accuracy: 0.9551
Accuracy:  0.9550955295562744
```

RNN

```
from tensorflow.keras import preprocessing
```

```
max_features  = vocab_size
max_len = 50
```

```
train_data = preprocessing.sequence.pad_sequences(X_train, maxlen=max_len)
test_data = preprocessing.sequence.pad_sequences(X_test, maxlen=max_len )
```

```
clear_session()
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.SimpleRNN(32))
model.add(layers.Dense(1, activation='sigmoid')) # sigmoid for binary classification
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, None, 32)          320000    
                                                                 
 simple_rnn (SimpleRNN)      (None, 32)                2080      
                                                                 
 dense (Dense)               (None, 1)                 33        
                                                                 
=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
_________________________________________________________________
```

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

```
history = model.fit(train_data, y_train, epochs=20, batch_size=128, validation_split=0.2)
```

```
Epoch 1/20
79/79 [==============================] - 4s 28ms/step - loss: 0.6848 - accuracy: 0.5652 - val_loss: 0.6853 - val_accuracy: 0.5621
Epoch 2/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6837 - accuracy: 0.5688 - val_loss: 0.6855 - val_accuracy: 0.5617
Epoch 3/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6837 - accuracy: 0.5691 - val_loss: 0.6853 - val_accuracy: 0.5637
Epoch 4/20
79/79 [==============================] - 3s 38ms/step - loss: 0.6831 - accuracy: 0.5712 - val_loss: 0.6865 - val_accuracy: 0.5637
Epoch 5/20
79/79 [==============================] - 2s 31ms/step - loss: 0.6836 - accuracy: 0.5713 - val_loss: 0.6853 - val_accuracy: 0.5621
Epoch 6/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6832 - accuracy: 0.5707 - val_loss: 0.6858 - val_accuracy: 0.5653
Epoch 7/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6834 - accuracy: 0.5711 - val_loss: 0.6857 - val_accuracy: 0.5625
Epoch 8/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6829 - accuracy: 0.5718 - val_loss: 0.6849 - val_accuracy: 0.5637
Epoch 9/20
79/79 [==============================] - 2s 22ms/step - loss: 0.6827 - accuracy: 0.5715 - val_loss: 0.6849 - val_accuracy: 0.5641
Epoch 10/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6832 - accuracy: 0.5736 - val_loss: 0.6855 - val_accuracy: 0.5625
Epoch 11/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6825 - accuracy: 0.5716 - val_loss: 0.6852 - val_accuracy: 0.5633
Epoch 12/20
79/79 [==============================] - 2s 24ms/step - loss: 0.6829 - accuracy: 0.5726 - val_loss: 0.6846 - val_accuracy: 0.5629
Epoch 13/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6828 - accuracy: 0.5730 - val_loss: 0.6847 - val_accuracy: 0.5629
Epoch 14/20
```

```
79/79 [==============================] - 2s 23ms/step - loss: 0.6823 - accuracy: 0.5725 - val_loss: 0.6885 - val_accuracy: 0.5629
Epoch 15/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6827 - accuracy: 0.5728 - val_loss: 0.6876 - val_accuracy: 0.5641
Epoch 16/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6828 - accuracy: 0.5717 - val_loss: 0.6851 - val_accuracy: 0.5621
Epoch 17/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6821 - accuracy: 0.5728 - val_loss: 0.6847 - val_accuracy: 0.5637
Epoch 18/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6826 - accuracy: 0.5727 - val_loss: 0.6855 - val_accuracy: 0.5633
Epoch 19/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6830 - accuracy: 0.5719 - val_loss: 0.6843 - val_accuracy: 0.5649
Epoch 20/20
79/79 [==============================] - 2s 23ms/step - loss: 0.6827 - accuracy: 0.5726 - val_loss: 0.6846 - val_accuracy: 0.5641
```

```
from sklearn.metrics import classification_report

pred = model.predict(test_data)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
99/99 [==============================] - 1s 4ms/step
              precision    recall  f1-score   support

         0.0       0.67      0.01      0.03      1391
         1.0       0.56      0.99      0.72      1749

    accuracy                           0.56      3140
   macro avg       0.61      0.50      0.37      3140
weighted avg       0.61      0.56      0.41      3140
```

The results are pretty sad. RNN took quite a long time to train compared to the sequential model and the accuracy is much lower at 56%. This could be due the vanishing gradient problem, so I want to build another model.

I want to retry with LSTM and see if that helps.

```
clear_session()
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.LSTM(32))
model.add(layers.Dense(1, activation='sigmoid')) # sigmoid for binary classification
```

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

```
history = model.fit(train_data, y_train, epochs=10, batch_size=128, validation_split=0.2) # fewer epochs because slower
```

```
Epoch 1/10
79/79 [==============================] - 7s 53ms/step - loss: 0.6857 - accuracy: 0.5655 - val_loss: 0.6862 - val_accuracy: 0.5621
Epoch 2/10
79/79 [==============================] - 4s 45ms/step - loss: 0.6838 - accuracy: 0.5686 - val_loss: 0.6856 - val_accuracy: 0.5621
Epoch 3/10
79/79 [==============================] - 6s 70ms/step - loss: 0.6838 - accuracy: 0.5686 - val_loss: 0.6859 - val_accuracy: 0.5621
Epoch 4/10
79/79 [==============================] - 6s 72ms/step - loss: 0.6841 - accuracy: 0.5686 - val_loss: 0.6854 - val_accuracy: 0.5621
Epoch 5/10
79/79 [==============================] - 4s 46ms/step - loss: 0.6838 - accuracy: 0.5686 - val_loss: 0.6855 - val_accuracy: 0.5621
Epoch 6/10
79/79 [==============================] - 4s 45ms/step - loss: 0.6839 - accuracy: 0.5686 - val_loss: 0.6855 - val_accuracy: 0.5621
Epoch 7/10
79/79 [==============================] - 4s 45ms/step - loss: 0.6837 - accuracy: 0.5686 - val_loss: 0.6857 - val_accuracy: 0.5621
Epoch 8/10
79/79 [==============================] - 4s 46ms/step - loss: 0.6839 - accuracy: 0.5686 - val_loss: 0.6854 - val_accuracy: 0.5621
Epoch 9/10
79/79 [==============================] - 4s 45ms/step - loss: 0.6838 - accuracy: 0.5686 - val_loss: 0.6854 - val_accuracy: 0.5621
Epoch 10/10
79/79 [==============================] - 4s 45ms/step - loss: 0.6839 - accuracy: 0.5686 - val_loss: 0.6854 - val_accuracy: 0.5621
```

```
pred = model.predict(test_data)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
99/99 [==============================] - 1s 7ms/step
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00      1391
         1.0       0.56      1.00      0.72      1749

    accuracy                           0.56      3140
   macro avg       0.28      0.50      0.36      3140
weighted avg       0.31      0.56      0.40      3140

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in lab
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in lab
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in lab
  _warn_prf(average, modifier, msg_start, len(result))
```

The results are still extremely unimpressive.

Embedding Trial

```
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
from tensorflow import keras
```

```
embedding_dim = 128
max_seq = 200
```

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

```
voc = vectorizer.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))
```

```
# set up embedding layer
embedding_layer = layers.Embedding(len(word_index) + 1, embedding_dim, input_length=max_seq)
```

```
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="sigmoid")(x)
model = keras.Model(int_sequences_input, preds)
model.summary()

```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, None)]            0

 embedding_1 (Embedding)     (None, None, 128)         2560128

 conv1d_9 (Conv1D)           (None, None, 128)         82048

 max_pooling1d_6 (MaxPooling  (None, None, 128)        0
 1D)

 conv1d_10 (Conv1D)          (None, None, 128)         82048

 max_pooling1d_7 (MaxPooling  (None, None, 128)        0
 1D)

 conv1d_11 (Conv1D)          (None, None, 128)         82048

 global_max_pooling1d_3 (Glo  (None, 128)              0
 balMaxPooling1D)

 dense_7 (Dense)             (None, 128)               16512

 dropout_3 (Dropout)         (None, 128)               0

 dense_8 (Dense)             (None, 1)                 129

=================================================================
Total params: 2,822,913
Trainable params: 2,822,913
Non-trainable params: 0
_________________________________________________________________
```

```
X_train = vectorizer(np.array([[s] for s in train_text])).numpy()
X_test = vectorizer(np.array([[s] for s in test_text])).numpy()

y_train = np.array(train_labels)
y_test = np.array(test_labels)
```

```
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
model.fit(X_train, y_train, batch_size=128, epochs=10, validation_data=(X_test, y_test))
```

```
Epoch 1/10
99/99 [==============================] - 51s 506ms/step - loss: 0.0590 - accuracy: 0.9806 - val_loss: 0.1301 - val_accuracy: 0.9548
Epoch 2/10
99/99 [==============================] - 46s 470ms/step - loss: 0.0087 - accuracy: 0.9981 - val_loss: 0.1788 - val_accuracy: 0.9541
Epoch 3/10
99/99 [==============================] - 46s 466ms/step - loss: 7.9227e-04 - accuracy: 0.9998 - val_loss: 0.2363 - val_accuracy: 0.9529
Epoch 4/10
99/99 [==============================] - 45s 457ms/step - loss: 9.1496e-05 - accuracy: 1.0000 - val_loss: 0.2453 - val_accuracy: 0.9535
Epoch 5/10
99/99 [==============================] - 65s 657ms/step - loss: 4.2321e-05 - accuracy: 1.0000 - val_loss: 0.2626 - val_accuracy: 0.9532
Epoch 6/10
99/99 [==============================] - 56s 562ms/step - loss: 2.7528e-05 - accuracy: 1.0000 - val_loss: 0.2768 - val_accuracy: 0.9529
Epoch 7/10
99/99 [==============================] - 49s 494ms/step - loss: 1.2811e-05 - accuracy: 1.0000 - val_loss: 0.2927 - val_accuracy: 0.9532
Epoch 8/10
```

```
99/99 [==============================] - 58s 592ms/step - loss: 1.4771e-05 - accuracy: 1.0000 - val_loss: 0.3053 - val_accuracy: 0.9529
Epoch 9/10
99/99 [==============================] - 49s 498ms/step - loss: 7.3442e-06 - accuracy: 1.0000 - val_loss: 0.3139 - val_accuracy: 0.9525
Epoch 10/10
99/99 [==============================] - 59s 601ms/step - loss: 5.9494e-06 - accuracy: 1.0000 - val_loss: 0.3210 - val_accuracy: 0.9525
<keras.callbacks.History at 0x7fcc53366520>
```

Analysis

I was initially very excited about my dataset choice. I was excited at the thought of determining genre based upon a script snippet. I played with the original data for a long time before determining that 7 potential categories were too many for me to overcome as a beginner. The highest accuracy I could muster was about 16% after mulitple hours. So I narrowed the potential genres to only drama and thriller. I preprocessed my text to be normalized and remove stop words and some of the nonsense words.

Now that multinomial classificaiton problem was a binary classificaiton problem, I felt more confident. Upon my first sequnential model run, I acheived over 90% accuracy. I used a sigmoid output node activiation for the binary output expected. I changed the optimizer to Adam since I didn't use One Hot encoding, and used binary-cross entropy for loss and acheived a max of 95%.

My next model was going to be a simple RNN. I had to preprocess the text slightly differently in this case. I kept the same preprocessing from before but added padding so everything was the same length. This model took much longer to run and gave horrendous accuracy. I then tried to overcome any potential vanishing gradient issues by using an LSTM. The accuracy for this was also disappointing.

The last model I built dealt with embeddings. The embedding process felt very complicated and the model took seemingly forever to run (this could be due to a slow internet connection), but the accuracy is pretty good at 95% and it learned very fast. I don't know if it is worth all of that fuss however for comparable results to the oringinal sequential model.

✓ 10s completed at 5:52 PM