```
import pandas as pd
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
import numpy as np
```

```
# Load dataset. This dataset is available at https://www.kaggle.com/datasets/datasnaek/mbti
# The full details of the dataset are available on Kaggle.
# The dataset contains a Myers-Briggs Personality type, which will serve as the target,
# and a collection of parts of their 50 most recent posts, which serve as the text input.
# I will split up the text and maintain the personality type to make the model more general
# applicable.
df = pd.read_csv('mbti_1.csv', dtype={'posts':str, 'type':str})
```

```
df.head()
```

	type	posts
0	INFJ	'http://www.youtube.com/watch?v=qsXHcwe3krw|||...
1	ENTP	'I'm finding the lack of me in these posts ver...
2	INTP	'Good one _____ https://www.youtube.com/wat...
3	INTJ	'Dear INTP, I enjoyed our conversation the o...
4	ENTJ	'You're fired.|||That's another silly misconce...

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8675 entries, 0 to 8674
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   type    8675 non-null   object
 1   posts   8675 non-null   object
dtypes: object(2)
memory usage: 135.7+ KB
```

```
df['type'].unique()
```

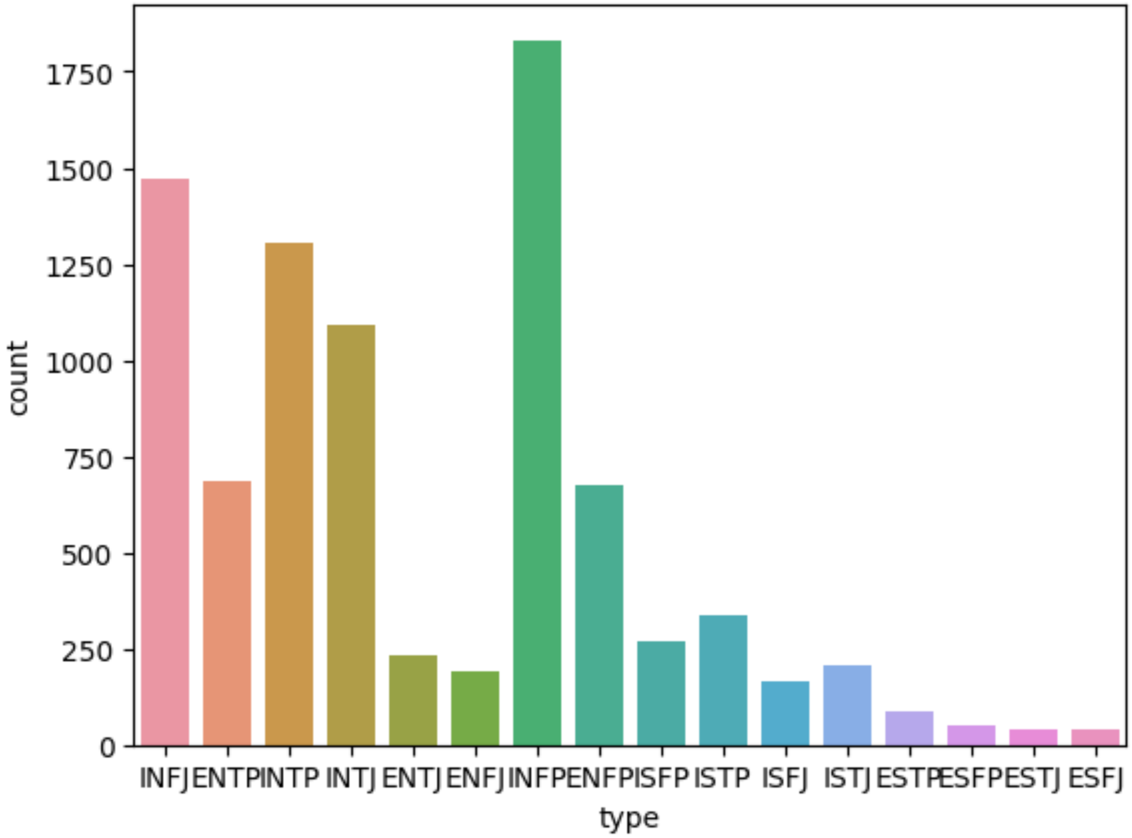
```
array(['INFJ', 'ENTP', 'INTP', 'INTJ', 'ENTJ', 'ENFJ', 'INFP', 'ENFP',
       'ISFP', 'ISTP', 'ISFJ', 'ISTJ', 'ESTP', 'ESFP', 'ESTJ', 'ESFJ'],
      dtype=object)
```

```
# Some personality types are more well-represented than others
df['type'].value_counts()
```

```
INFP    1832
INFJ    1470
INTP    1304
INTJ    1091
ENTP     685
ENFP     675
ISTP     337
ISFP     271
ENTJ     231
ISTJ     205
ENFJ     190
ISFJ     166
ESTP      89
ESFP      48
ESFJ      42
ESTJ      39
Name: type, dtype: int64
```

```
# Graph the distribution of target classes (equal amounts of Kecimen and Besni species)
sns.countplot(x=df['type'])
```

```
<AxesSubplot: xlabel='type', ylabel='count'>
```



```
# Convert compiled texts into lists
df['text'] = df['posts'].apply(lambda e: e.split('|||'))  # for now just remove the |||
df.drop('posts', axis=1, inplace=True)
df['text'].head()
```

```
0    ['http://www.youtube.com/watch?v=qsXHcwe3krw, ...
1    ['I'm finding the lack of me in these posts ve...
2    ['Good one  _____   https://www.youtube.com/wa...
3    ['Dear INTP,   I enjoyed our conversation the ...
4    ['You're fired., That's another silly misconce...
Name: text, dtype: object
```

```
# Convert lists into individual rows; 1 row -> 5 rows, and stop associating them by ignori
df = df.explode('text', ignore_index=True)
df.head()
```

	type	text
0	INFJ	'http://www.youtube.com/watch?v=qsXHcwe3krw
1	INFJ	http://41.media.tumblr.com/tumblr_lfouy03PMA1q...
2	INFJ	enfp and intj moments https://www.youtube.com...
3	INFJ	What has been the most life-changing experienc...
4	INFJ	http://www.youtube.com/watch?v=vXZeYwwRDw8 h...

```
# The dataset is enormous now
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 422845 entries, 0 to 422844
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   type    422845 non-null  object
 1   text    422845 non-null  object
dtypes: object(2)
memory usage: 6.5+ MB
```

```
# Let's trim it down.
df = df.truncate(after=10000)
```

```
# Split the DataFrame into features and target
X = df['text']
y = df['type']
```

```
# Create a dictionary to convert classes to numbers for the model
enumerated_classes = {key: i for i, key in enumerate(y.unique())}
enumerated_classes
```

```
{'INFJ': 0,
    {'INFJ': 0,
     'ENTP': 1,
     'INTP': 2,
     'INTJ': 3,
     'ENTJ': 4,
     'ENFJ': 5,
     'INFP': 6,
     'ENFP': 7,
     'ISFP': 8,
     'ISTP': 9,
     'ISFJ': 10,
     'ISTJ': 11,
     'ESTP': 12,
     'ESFP': 13}
```

```
# Convert y to a list of numeric values
y = y.apply(lambda type_name: enumerated_classes[type_name]
    )
```

```
# Divide the data into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
# Ensure we chose the right variables from the train test split
len(X_train) == len(y_train) and len(X_test) == len(y_test)
```

```
    True
```

```
# Vectorize the text
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=400) # Only get top 400 terms to conserve memory
tokenizer.fit_on_texts(X_train)
```

```
# Vectorized data
X_train_vectorized = tokenizer.texts_to_sequences(X_train)
X_test_vectorized = tokenizer.texts_to_sequences(X_test)
```

```
X_train_vectorized
```

```
    [[223, 16, 40, 160, 13, 1, 226, 265, 5, 24, 26, 1, 287, 57],
     [2, 371, 15, 55, 33, 1, 78, 43, 2, 314, 1, 287, 4, 14, 198, 61, 1, 3, 2],
     [1,
      1,
      124,
      14,
      5,
      195,
      61,
      256,
  3,
  244,
  124,
  198,
  191,
  90,
  1,
  32,
  339,
  3,
  67,
  21,
  1,
  219,
  126,
  3,
  244,
  124,
  164,
  5,
  158,
  90,
  44,
  191,
  9,
  353,
  1],
 [1, 36, 51, 22, 51, 42, 29, 47, 82, 99, 74, 1, 223, 16, 40, 216],
 [19, 3, 2, 39, 26, 162],
 [],
 [1,
  31,
  280,
  4,
  18,
  232,
  8,
  2,
  34,
  40,
  60,
  8,
  59,
  207,
  20,
  323,
  4,
  74,
```

```
# Pad the sequences so that they are the same length
from tensorflow.keras.preprocessing.sequence import pad_sequences
X_train_vectorized = pad_sequences(X_train_vectorized, maxlen=50, padding='post', truncatin
X_test_vectorized = pad_sequences(X_test_vectorized, maxlen=50, padding='post', truncating=
X_train_vectorized
```

```
array([[223.,  16.,  40., ...,   0.,   0.,   0.],
       [  2., 371.,  15., ...,   0.,   0.,   0.],
       [  1.,   1., 124., ...,   0.,   0.,   0.],
       ...,
       [341.,  51.,  22., ...,   0.,   0.,   0.],
       [ 33.,  29.,   7., ...,   0.,   0.,   0.],
       [  1., 398., 106., ...,   0.,   0.,   0.]], dtype=float32)
```

```
seq_model = keras.Sequential([
    keras.layers.Embedding(400, 32, input_length=50),
    keras.layers.LSTM(64, dropout=0.1),
    keras.layers.Dense(len(enumerated_classes), activation='softmax')
])
```

```
seq_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 32)            12800

 lstm (LSTM)                 (None, 64)                24832

 dense (Dense)               (None, 14)                910

=================================================================
Total params: 38,542
Trainable params: 38,542
Non-trainable params: 0
_________________________________________________________________
```

```
seq_model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

```
# seq_model.fit(X_train_vectorized, y_train, epochs=5)
# My PC completely crashes and restarts a couple seconds after running this so... I won't.
# Instead I will continue with the rest of the code but will not be able to run it.
```

```
Epoch 1/5
250/250 [==============================] - 2s 7ms/step - loss: 2.2231 - accuracy: 0.1
Epoch 2/5
244/250 [============================>.] - ETA: 0s - loss: 2.2221 - accuracy: 0.1820
```

```
# Check for overfitting by scoring against test values
```

```
seq_model.evaluate(X_test_vectorized, y_test)
```

```
63/63 [==============================] - 0s 4ms/step - loss: 2.2522 - accuracy: 0.170
[2.2521889209747314, 0.17091454565525055]
```

```
# Create a RNN
rnn = keras.Sequential([
    keras.layers.Embedding(input_dim=400, output_dim=60),
    keras.layers.LSTM(100),
    keras.layers.Dense(len(enumerated_classes)) # Output layer
])

rnn.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, None, 60)          24000

 lstm_2 (LSTM)               (None, 100)               64400

 dense_2 (Dense)             (None, 14)                1414

=================================================================
Total params: 89,814
Trainable params: 89,814
Non-trainable params: 0
_________________________________________________________________
```

```
rnn.evaluate(X_test_vectorized, y_test)
```

```
# Trying different embedding approaches
# Instead of assigning each word an integer and listing them in sequence, assign each sente
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(lowercase=False, max_df=0.90, min_df=0.01) # Ignore terms that
```

```
vectorizer.fit(X_train)

# Vectorized data
X_train_vectorized = vectorizer.transform(X_train).toarray()
X_test_vectorized = vectorizer.transform(X_test).toarray()
```

```
X_train_vectorized = pad_sequences(X_train_vectorized, maxlen=50, padding='post', truncatin
X_test_vectorized = pad_sequences(X_test_vectorized, maxlen=50, padding='post', truncating=
```

```
seq_model = keras.Sequential([
    keras.layers.Embedding(400, 32, input_length=50),
    keras.layers.LSTM(64, dropout=0.1),
    keras.layers.Dense(len(enumerated_classes), activation='softmax')
])

seq_model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)


seq_model.fit(X_train_vectorized, y_train, epochs=5)


seq_model.evaluate(X_test_vectorized, y_test)
```

```
63/63 [==============================] - 1s 3ms/step - loss: 2.2475 - accuracy: 0.193
[2.247462034225464, 0.19340330362319946]
```

The tokenizing approach where each (top) word is assigned an integer value is likely to perform much better than the bag-of-words style vectorization because it encodes within it the sequential data of the text that is lost when doing a bag-of-words. Unfortunately I could not run all the models to verify the difference in performance due to my poor setup or whatever the cause of my computer crashing is if not that. Other existing pre-trained models like BERT could serve as alternative embeddings which would likely improve the performance of this text classifier.