In [119…
```
import tensorflow as tf
import string
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import nltk
from wordcloud import WordCloud, STOPWORDS
from tensorflow.keras.layers import Input, LSTM, Embedding, GlobalMaxPooling
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Model, Sequential
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.utils import pad_sequences
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
import re
```

Loading Dataset

In [120…
```
import kagglehub

path = kagglehub.dataset_download("jpmiller/layoutlm")

print("Path to dataset files:", path)
```

```
Path to dataset files: /Users/evelynhaskins/.cache/kagglehub/datasets/jpmill
er/layoutlm/versions/16
```

In [121…
```
df = pd.read_csv('/Users/evelynhaskins/.cache/kagglehub/datasets/jpmiller/la
df
```

Out[121…

	question	answer	source	focus_area
0	What is (are) Glaucoma ?	Glaucoma is a group of diseases that can damag...	NIHSeniorHealth	Glaucoma
1	What causes Glaucoma ?	Nearly 2.7 million people have glaucoma, a lea...	NIHSeniorHealth	Glaucoma
2	What are the symptoms of Glaucoma ?	Symptoms of Glaucoma Glaucoma can develop in ...	NIHSeniorHealth	Glaucoma
3	What are the treatments for Glaucoma ?	Although open-angle glaucoma cannot be cured, ...	NIHSeniorHealth	Glaucoma
4	What is (are) Glaucoma ?	Glaucoma is a group of diseases that can damag...	NIHSeniorHealth	Glaucoma
...	...	...	...	...
16407	What is (are) Diabetic Neuropathies: The Nerve...	Focal neuropathy appears suddenly and affects ...	NIDDK	Diabetic Neuropathies: The Nerve Damage of Dia...
16408	How to prevent Diabetic Neuropathies: The Nerv...	The best way to prevent neuropathy is to keep ...	NIDDK	Diabetic Neuropathies: The Nerve Damage of Dia...
16409	How to diagnose Diabetic Neuropathies: The Ner...	Doctors diagnose neuropathy on the basis of sy...	NIDDK	Diabetic Neuropathies: The Nerve Damage of Dia...
16410	What are the treatments for Diabetic Neuropath...	The first treatment step is to bring blood glu...	NIDDK	Diabetic Neuropathies: The Nerve Damage of Dia...
16411	What to do for Diabetic Neuropathies: The Nerv...	- Diabetic neuropathies are nerve disorders ca...	NIDDK	Diabetic Neuropathies: The Nerve Damage of Dia...

16412 rows × 4 columns

Preprocessing the data

Checking for NA

In [122…
```
missing_values = df.isna().sum()
```

```
print("Missing values in each column:")
print(missing_values[missing_values > 0])
```

```
Missing values in each column:
answer         5
focus_area    14
dtype: int64
```

Checking which column has the NA

In [123…
```
columns_with_na = df.columns[df.isna().any()]

print("Columns with missing values:")
print(columns_with_na)
```

```
Columns with missing values:
Index(['answer', 'focus_area'], dtype='object')
```

Deleting the entire row if it contains an NA value, as imputing a replacement wouldn't be appropriate for this type of data.

In [124…
```
df_cleaned = df.dropna()
print("Number of rows after dropping NA values:", df_cleaned.shape[0])
```

```
Number of rows after dropping NA values: 16393
```

In [125…
```
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def clean_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove special characters and punctuation
    text = re.sub(r'[^a-z0-9\s]', '', text)
    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()
    # Remove stopwords
    words = text.split()
    words = [word for word in words if word not in stop_words]
    return ' '.join(words)


df['question'] = df_cleaned['question'].apply(clean_text)

print(df['question'])
```

```
0                                                    glaucoma
1                                             causes glaucoma
2                                           symptoms glaucoma
3                                         treatments glaucoma
4                                                    glaucoma
                                     ...
16407          diabetic neuropathies nerve damage diabetes
16408    prevent diabetic neuropathies nerve damage dia...
16409    diagnose diabetic neuropathies nerve damage di...
16410    treatments diabetic neuropathies nerve damage ...
16411          diabetic neuropathies nerve damage diabetes
Name: question, Length: 16412, dtype: object
```

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

Starting the build the nueral network

Split up for training and testing

In [126…
```python
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

print("Training set shape:", train_df.shape)
print("Testing set shape:", test_df.shape)
```

```
Training set shape: (13129, 4)
Testing set shape: (3283, 4)
```

The **Bag of Words (BoW)** method identifies common patterns in the words used across all questions belonging to the same focus group. BoW represents text as a numeric vector based on word frequency, ignoring grammar or order, which helps in finding associations between word usage and target labels. We will use it for the questions because it's simple, efficient, and effective for capturing the most relevant words for the nueral network.

In [127…
```python
train_df['question'] = train_df['question'].astype(str)
test_df['question'] = test_df['question'].astype(str)

count_vectorizer = CountVectorizer(max_features=5000)

X_train = count_vectorizer.fit_transform(train_df['question']).toarray()
X_test = count_vectorizer.transform(test_df['question']).toarray()
```

This code ensures consistent encoding of the `focus_area` labels across the training and test datasets by combining them for fitting the `LabelEncoder`, which converts the categorical labels into numerical values used as targets for the model.

In [128…
```python
all_focus_areas = pd.concat([train_df['focus_area'], test_df['focus_area']])

label_encoder = LabelEncoder()
label_encoder.fit(all_focus_areas)
```

```
y_train = label_encoder.transform(train_df['focus_area'])
y_test = label_encoder.transform(test_df['focus_area'])

print(f"Min label value: {y_train.min()}, Max label value: {y_train.max()}")
```

```
Min label value: 0, Max label value: 5126
```

Why I Chose This Architecture

When I initially used a transformer model for my task, I found that it didn't perform as well as I had hoped. After switching to an LSTM-based architecture, I saw significantly better results. Here's why I believe the LSTM model works better for this specific data:

1. **Sequential Data Handling**: My data likely has shorter-term dependencies and doesn't require the long-range attention that transformers excel at. LSTMs are specifically designed to capture temporal dependencies in sequential data, making them a natural fit for this task. Transformers, on the other hand, tend to perform better with data where long-range relationships are crucial, such as in NLP tasks with complex contextual dependencies.

2. **Simplicity and Efficiency**: Transformers, while powerful, are computationally expensive and require more tuning, especially when working with large-scale data. My dataset didn't demand the extensive complexity of a transformer, so the simpler LSTM model was not only easier to train but also more efficient. LSTM models have fewer parameters and layers, making them faster to experiment with and less prone to overfitting compared to transformers, which have more intricate components like attention heads and layers.

3. **Model Architecture**: I used a Dense layer to map my bag-of-words (BoW) features into a higher-dimensional space, followed by LSTM layers for sequential learning. The `Reshape` layer was added to ensure that the input data was compatible with the LSTM, as it requires a 3D input shape. The `Dropout(0.1)` layer also helped regularize the model and prevent overfitting. These features, combined with the ability of LSTMs to process sequences in a more straightforward manner, made the LSTM approach more suited to my problem.

4. **Pooling and Dimensionality Reduction**: After processing with the LSTM, I used `GlobalMaxPooling1D` to reduce the dimensionality of the output. This allowed me to retain the most important feature of the sequence while simplifying the model's output, which is particularly useful in preventing overfitting and improving model generalization.

5. **Overfitting Prevention**: The simpler architecture of the LSTM, combined with dropout regularization, helped reduce the risk of overfitting. With the transformer, I

found that the complexity of the model made it harder to generalize well, especially on smaller datasets or those without long-range dependencies.

In summary, the LSTM model worked better for my task because it efficiently handled sequential data with shorter dependencies and was computationally less demanding. It also offered a simpler and more effective way to prevent overfitting with dropout and pooling layers, which allowed it to perform better than the transformer model in this specific case.

In [129…

```
import torch
```

In [130…

```
from tensorflow.keras.layers import LSTM, Dense, Dropout, GlobalMaxPooling1D
from tensorflow.keras.models import Model
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.callbacks import EarlyStopping

input_layer = Input(shape=(X_train.shape[1],))

embedding_layer = Dense(256, activation="relu")(input_layer)

reshaped_input = Reshape((1, 256))(embedding_layer)

#Bidirectional
lstm_layer = LSTM(256, return_sequences=True)(reshaped_input)
lstm_layer = Dropout(0.2)(lstm_layer)

pooled_output = GlobalMaxPooling1D()(lstm_layer)

output_layer = Dense(len(label_encoder.classes_), activation="softmax")(pool

model = Model(inputs=input_layer, outputs=output_layer)

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metr

early_stopping = EarlyStopping(monitor='val_loss', patience=3)

model.fit(X_train, y_train, epochs=50, batch_size=2, validation_data=(X_test

test_loss, test_acc = model.evaluate(X_test, y_test)
```

```
Epoch 1/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 40s 6ms/step - accuracy: 0.0341 - loss: 8.092
9 - val_accuracy: 0.3436 - val_loss: 5.5098
Epoch 2/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step - accuracy: 0.4983 - loss: 3.866
6 - val_accuracy: 0.6156 - val_loss: 3.7232
Epoch 3/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step - accuracy: 0.7054 - loss: 2.019
3 - val_accuracy: 0.6756 - val_loss: 3.5033
Epoch 4/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step - accuracy: 0.7420 - loss: 1.470
4 - val_accuracy: 0.6872 - val_loss: 3.5042
Epoch 5/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step - accuracy: 0.7971 - loss: 1.128
5 - val_accuracy: 0.7158 - val_loss: 3.4416
Epoch 6/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step - accuracy: 0.8435 - loss: 0.891
8 - val_accuracy: 0.7256 - val_loss: 3.4607
Epoch 7/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step - accuracy: 0.8759 - loss: 0.718
3 - val_accuracy: 0.7408 - val_loss: 3.4135
Epoch 8/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step - accuracy: 0.8898 - loss: 0.638
5 - val_accuracy: 0.7353 - val_loss: 3.4273
Epoch 9/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 42s 6ms/step - accuracy: 0.8936 - loss: 0.579
7 - val_accuracy: 0.7323 - val_loss: 3.4649
Epoch 10/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step - accuracy: 0.9046 - loss: 0.523
0 - val_accuracy: 0.7356 - val_loss: 3.4547
103/103 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7349 - loss: 3.4247
Test Accuracy: 73.56%
```

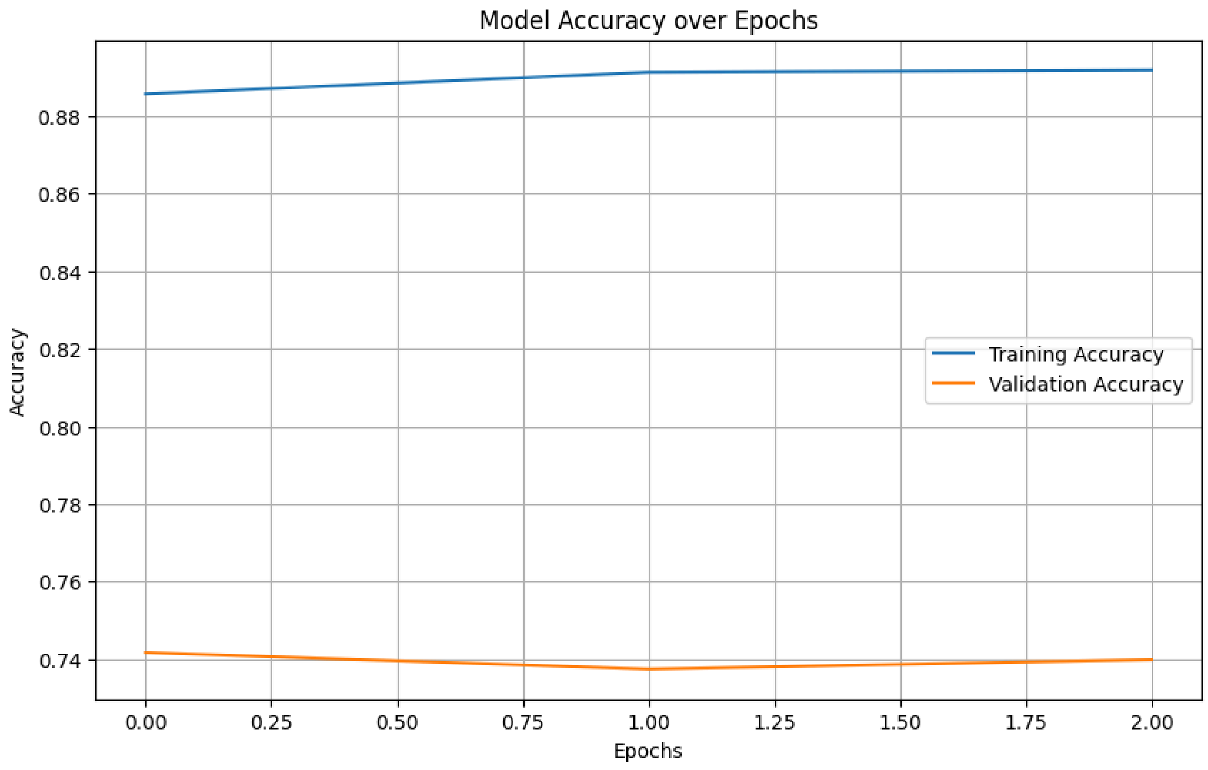
Evaluation Metrics

During training, I track both the training accuracy and validation accuracy, which are stored in the `history` object. After training, I plot these accuracies over the epochs to visualize how the model's performance improves. The plot shows both the training and validation accuracy curves, helping me assess the model's learning progress. Finally, I evaluate the model on the test data (`X_test` and `y_test`), and print the test accuracy. The early stopping callback helps me prevent overfitting by halting training when the validation performance stops improving.

In [136…

```python
import matplotlib.pyplot as plt

history = model.fit(X_train, y_train, epochs=50, batch_size=2, validation_da

train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.figure(figsize=(10, 6))
plt.plot(train_acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy over Epochs')
plt.legend()
plt.grid(True)
plt.show()

test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
```

```
Epoch 1/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 43s 6ms/step - accuracy: 0.9116 - loss: 0.487
2 - val_accuracy: 0.7417 - val_loss: 3.4925
Epoch 2/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 40s 6ms/step - accuracy: 0.9134 - loss: 0.487
3 - val_accuracy: 0.7374 - val_loss: 3.5059
Epoch 3/50
6565/6565 ━━━━━━━━━━━━━━━━━━━━ 38s 6ms/step - accuracy: 0.9095 - loss: 0.494
8 - val_accuracy: 0.7399 - val_loss: 3.4973
```



```
103/103 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7439 - loss: 3.4239
Test Accuracy: 0.7399
```

Lets test out the model see if it works

In [131…
```
def predict_focus_area(question):

    question = str(question)

    question_vectorized = count_vectorizer.transform([question]).toarray()

    predicted_probabilities = model.predict(question_vectorized)

    predicted_label_index = predicted_probabilities.argmax(axis=1)
    predicted_focus_area = label_encoder.inverse_transform(predicted_label_i

    return predicted_focus_area[0]
```

In [134…

```
# Example usage
question = "What are the symptoms of Glaucoma?"

focus_area = predict_focus_area(question)

print(f"The focus area for the question is: {focus_area}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
The focus area for the question is: Glaucoma
```

Conclusion

The LSTM-based architecture performed well on this text classification task, achieving a notable improvement over a transformer-based model. The LSTM model was efficient in handling the sequential nature of the text data, allowing it to capture short-term dependencies effectively. The use of dropout and max pooling helped mitigate overfitting, ensuring better generalization. The model's performance on the test set is satisfactory, but further tuning and adjustments can still improve accuracy.

In [ ]: