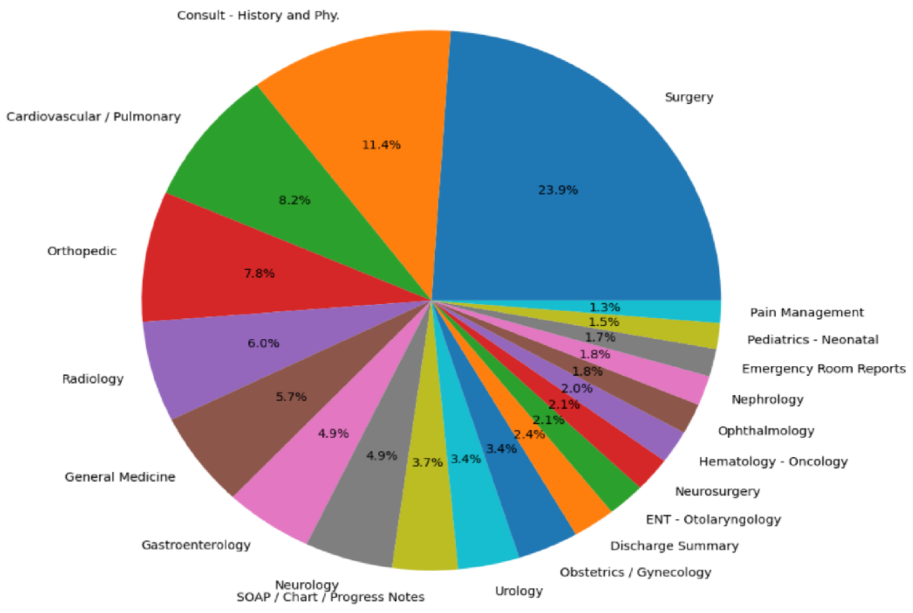
Dataset:

The dataset used in this study consists of two columns named "transcription" and "medical specialty." The objective of this research is to predict the "medical specialty" column using the text in the "transcription" column.

To achieve this goal, the dataset was examined and prepared in various ways, which will be explained later.

The "medical specialty" column has 40 classes, and the distribution of its various types is displayed in the following figure.

```
Surgery                           1103
Consult - History and Phy.         516
Cardiovascular / Pulmonary         372
Orthopedic                         355
Radiology                          273
General Medicine                   259
Gastroenterology                   230
Neurology                          223
SOAP / Chart / Progress Notes      166
Obstetrics / Gynecology            160
Urology                            158
Discharge Summary                  108
ENT - Otolaryngology                98
Neurosurgery                        94
Hematology - Oncology               90
Ophthalmology                       83
Nephrology                          81
Emergency Room Reports              75
Pediatrics - Neonatal               70
Pain Management                     62
Psychiatry / Psychology             53
Office Notes                        51
Podiatry                            47
Dermatology                         29
Cosmetic / Plastic Surgery          27
Dentistry                           27
Letters                             23
Physical Medicine - Rehab           21
Sleep Medicine                      20
Endocrinology                       19
Bariatrics                          18
IME-QME-Work Comp etc.              16
Chiropractic                        14
Rheumatology                        10
Diets and Nutritions                10
Speech - Language                    9
Autopsy                              8
Lab Medicine - Pathology             8
Allergy / Immunology                 7
Hospice - Palliative Care            6
```



As observed, the different classes in the "medical specialty" column are unevenly distributed. Additionally, the dataset consists of 4,999 records, and there are some missing values in the "transcription" column, resulting in 4,966 records remaining in this column. Given the defined task, records with missing values in this column were removed.

Given the label imbalance issue explained earlier, classification tasks face challenges. Therefore, as part of the preprocessing, the impact of reducing the number of labels to 13, including labels with a minimum of 100 records, was investigated.

Additionally, given that the goal is text classification, as another part of dataset preparation, text preprocessing for the "transcription" column was examined, which will be explained further below.

```
import pandas as pd
import string
import nltk
```

```
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer

nltk.download('stopwords')
nltk.download('wordnet')

def preprocess_column(df, column_name):
    # Remove punctuation
    def remove_punctuation(text):
        j=" "
        punctuation_free =  "".join([" " if i in string.punctuation else i for i
in text]) #"".join([i for i in text if i not in string.punctuation ])
        return punctuation_free

    df_copy = df.copy()
    df_copy['clean_' + column_name] = df_copy[column_name].apply(lambda x:
remove_punctuation(x))

    # Convert to lowercase
    df_copy['clean_' + column_name] = df_copy['clean_' +
column_name].apply(lambda x: x.lower())

    # Remove stopwords
    stop_words_list = stopwords.words('english')
    def remove_stopwords(text):
        words = text.split()
        meaningful_words = [word for word in words if word.lower() not in
stop_words_list]
        return " ".join(meaningful_words)

    df_copy['clean_' + column_name] = df_copy['clean_' +
column_name].apply(remove_stopwords)

    # Apply stemming
    porter_stemmer = PorterStemmer()
    def stemming(text):
        words = text.split()
        stem_text = [porter_stemmer.stem(word) for word in words]
        return " ".join(stem_text)

    df_copy['stemmed_' + column_name] = df_copy['clean_' +
column_name].apply(stemming)

    # Apply lemmatization
    wordnet_lemmatizer = WordNetLemmatizer()
    def lemmatizer(text):
        words = text.split()
        lemm_text = [wordnet_lemmatizer.lemmatize(word) for word in words]
        return " ".join(lemm_text)

    df_copy['lemmatized_' + column_name] = df_copy['clean_' +
column_name].apply(lemmatizer)

    return df_copy
```

The provided code serves the purpose of text preprocessing, and the following steps have been executed:

Step 1: Removing Punctuation In the "remove_punctuation" function, punctuation marks are eliminated from the text. This operation is accomplished through a list comprehension and the utilization of the "string.punctuation" module, which replaces punctuation characters with spaces.

Step 2: Converting to Lowercase Following the removal of punctuation, the text is transformed into lowercase letters by utilizing the ".lower()" method. This process ensures uniform casing throughout the text.

Step 3: Removing Stopwords Stopwords, which encompass common words like "the," "and," "in," and others that lack significant informational value, are excluded from the text. The "remove_stopwords" function dissects the text into tokens, eliminates stopwords using a list comprehension and the NLTK "stopwords.words('english')" list, and subsequently reconstructs the sentence with the remaining words.

Step 4: Stemming Stemming, a technique aimed at reducing words to their base or root form, is applied in this code using the Porter Stemmer from NLTK. The text is divided into words, stemming is applied to each word, and the words are then reunited to form a sentence.

Step 5: Lemmatization Lemmatization, a more sophisticated method for reducing words to their base forms (lemmas) while considering the word's context, is employed here. The code employs the WordNet Lemmatizer from NLTK to tokenize the text, lemmatize each word, and reassemble them into a sentence.

The final result is a DataFrame with various columns containing different versions of the preprocessed text. These include one with punctuation removed and converted to lowercase, one with stopwords removed, one with stemming applied, and one with lemmatization applied.

These preprocessing stages serve the purpose of preparing text data for subsequent analyses, such as text classification. They aid in reducing noise and standardizing the text, facilitating the work of machine learning algorithms with the data.

Therefore, the pipeline is divided into two states for text examination: preprocessed and unprocessed. It is also categorized based on the number of labels into two states: with 40 labels and with 13 labels. In the latter state, only labels with more than 100 records are retained. These labels include:

1. "Surgery"
2. "Consult - History and Phy."
3. "Cardiovascular / Pulmonary"
4. "Orthopedic"
5. "Radiology"
6. "General Medicine"
7. "Gastroenterology"
8. "Neurology"
9. "SOAP / Chart / Progress Notes"
10. "Obstetrics / Gynecology"
11. "Urology"
12. "Discharge Summary"
13. "Others"

Models:

Given that the task at hand is text classification, several models suitable for text processing and classification tasks were investigated. As a baseline, a sequential neural network model, which is well-suited for text classification tasks, was considered. In the comparative analysis, traditional machine learning models, known for their effectiveness in classification tasks, especially when dealing with limited data, were also examined. Additionally, several transformer-based models, including BERT and GPT2, were scrutinized.

Here's a brief summary of the models considered for the task:

1. **Sequential Neural Network Model (e.g., LSTM):**
   - Sequential neural network models are commonly used for text classification tasks due to their ability to capture sequential dependencies in text data.
2. **Traditional Machine Learning Models (e.g., Logistic Regression, Support Vector Machine):**
   - Traditional machine learning models were explored as powerful alternatives, especially in scenarios with limited data availability.
3. **Transformer-Based Models (e.g., BERT, GPT2):**
   - Transformer-based models, such as BERT and GPT2, were also evaluated. These models have achieved state-of-the-art performance in various natural language processing tasks and are known for their ability to handle contextual information effectively.

The purpose of comparing these models is to identify the most suitable one for the text classification task, taking into consideration factors like model performance, computational resources, and the amount of available data. Each model has its strengths and weaknesses, and the choice of model ultimately depends on the specific requirements and constraints of the task at hand.

To utilize traditional models for text classification, it is essential to first extract features from the text and then perform classification based on these features. Several methods were employed to extract necessary features from the text for two traditional models, SVM and Logistic Regression. Here, I'll explain three methods used:

1. TF-IDF (Term Frequency-Inverse Document Frequency):

- The TF-IDF method is employed to convert text into feature vectors that capture the importance of terms in documents.
- The code snippet below demonstrates how to implement TF-IDF:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=1, max_df=0.8, sublinear_tf=True,
use_idf=True)
train_vectors = vectorizer.fit_transform(train['text'])
test_vectors = vectorizer.transform(test['text'])
```

2. Count Vectorization:

- Count Vectorization converts text into a matrix of token counts. Each row represents a document, and each column represents a unique word in the entire corpus.
- The code snippet below demonstrates how to perform Count Vectorization:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
features_train = vectorizer.fit_transform(train['text'])
features_test = vectorizer.transform(test['text'])
```

3. Word Embeddings (GloVe):

- Word Embeddings are dense vector representations of words that capture semantic relationships between words.
- **GloVe (Global Vectors for Word Representation):** Utilizes pre-trained GloVe word vectors to convert words into vector representations. Pre-trained GloVe word vectors are loaded and utilized for feature extraction:

```
glove_file = '/content/glove.6B.100d.txt'  # Adjust the filename as needed
word_vectors = {}
with open(glove_file, 'r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.array(values[1:], dtype='float32')
        word_vectors[word] = vector
```

This method transforms text data into numerical features that can be used as input for traditional machine learning models like SVM and Logistic Regression. The choice of feature extraction method depends on the specific requirements and performance characteristics of the task at hand.

Svm model:

After extracting features, the SVM model was employed for training and classification. The following code snippets illustrate the training and evaluation of this model:

```
# Create an SVM classifier
svm_classifier = SVC(kernel='linear')

# Fit the classifier on the training data
svm_classifier.fit(X_train_scaled, y_train)

# Predict labels for the test data
predicted_labels = svm_classifier.predict(X_test_scaled)

# Calculate accuracy on the test set
accuracy = accuracy_score(y_test, predicted_labels)

print("Accuracy of the SVM model on enhanced word embeddings:", accuracy)
```

This SVM model was evaluated using various types of extracted text features.

Logistic regression model:

After SVM, Logistic Regression was used as another traditional machine learning model. This model, despite having a more straightforward parameter space compared to SVM, was implemented as follows:

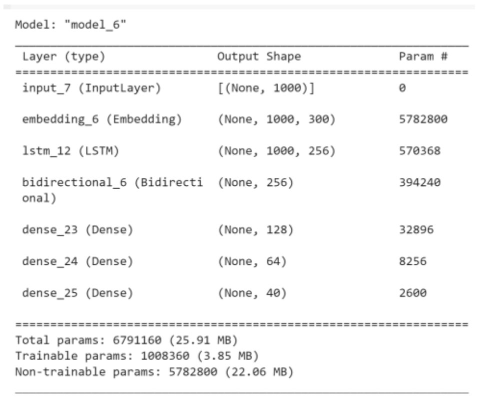
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Create a Logistic Regression classifier
logistic_classifier = LogisticRegression()

# Fit the classifier on the training data
logistic_classifier.fit(X_train_scaled, train['label'])
```

```
# Predict labels for the test data
predicted_labels = logistic_classifier.predict(X_test_scaled)

# Calculate accuracy on the test set
accuracy = accuracy_score(test['label'], predicted_labels)

print("Accuracy of the Logistic Regression model:", accuracy)
```

The code above demonstrates the implementation of the Logistic Regression model, which is known for its simplicity and ease of use. It was trained and evaluated using the extracted text features.

Deep Neural Network Models:

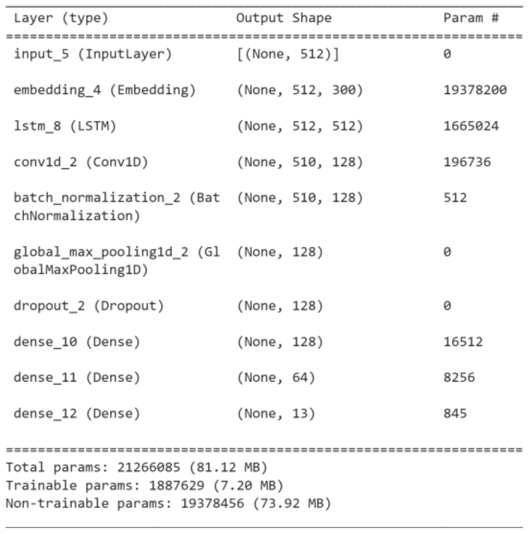
For the implementation of deep neural network models, especially for tasks involving sequences, both unidirectional and bidirectional LSTM models were employed. The reason for using these models lies in their ability to capture relationships between words in a sequence. Additionally, the use of convolutional layers, which are effective in feature identification and extraction, was incorporated. Furthermore, other parameters such as dropout, regularization, batch normalization, and more were utilized to enhance model outputs and prevent overfitting.



```
Model: "model_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 1000)]            0

 embedding_6 (Embedding)     (None, 1000, 300)         5782800

 lstm_12 (LSTM)              (None, 1000, 256)         570368

 bidirectional_6 (Bidirecti  (None, 256)               394240
 onal)

 dense_23 (Dense)            (None, 128)               32896

 dense_24 (Dense)            (None, 64)                8256

 dense_25 (Dense)            (None, 40)                2600

=================================================================
Total params: 6791160 (25.91 MB)
Trainable params: 1008360 (3.85 MB)
Non-trainable params: 5782800 (22.06 MB)
_________________________________________________________________
```

The image above illustrates the implemented model for the case without preprocessing with 40 classes. In this model, input is fed into the embedding layer, which uses pre-trained word2vec embeddings trained on the training dataset. Then, the output of the embedding layer is passed through an LSTM layer followed by a bidirectional LSTM (biLSTM) layer. Subsequently, it goes through two Dense layers, and finally, the output is generated using the softmax activation function. The same architecture is used again, but with different layer densities for the preprocessed case.

However, a more complex model was explored, leading to an improvement of nearly 1% in accuracy. In this model, the input is passed through an embedding layer, and then an LSTM layer is used to encode the input. In the next layer, a one-dimensional convolutional layer is employed to extract text features. Then, it passes through batch normalization, max pooling, and dropout layers to prevent overfitting. Finally, two dense layers are used for classification. This architecture is illustrated in the figure below.



```
Layer (type)                Output Shape              Param #
=================================================================
input_5 (InputLayer)        [(None, 512)]             0

embedding_4 (Embedding)     (None, 512, 300)          19378200

lstm_8 (LSTM)               (None, 512, 512)          1665024

conv1d_2 (Conv1D)           (None, 510, 128)          196736

batch_normalization_2 (Bat  (None, 510, 128)          512
chNormalization)

global_max_pooling1d_2 (Gl  (None, 128)               0
obalMaxPooling1D)

dropout_2 (Dropout)         (None, 128)               0

dense_10 (Dense)            (None, 128)               16512

dense_11 (Dense)            (None, 64)                8256

dense_12 (Dense)            (None, 13)                845

=================================================================
Total params: 21266085 (81.12 MB)
Trainable params: 1887629 (7.20 MB)
Non-trainable params: 19378456 (73.92 MB)
_________________________________________________________________
```

The results presented in the results section pertain to the first model, which utilizes LSTM and biLSTM.

Bert:

For the transformer models, PyTorch environment was utilized for implementation. Below is the code implementation for the BERT model:

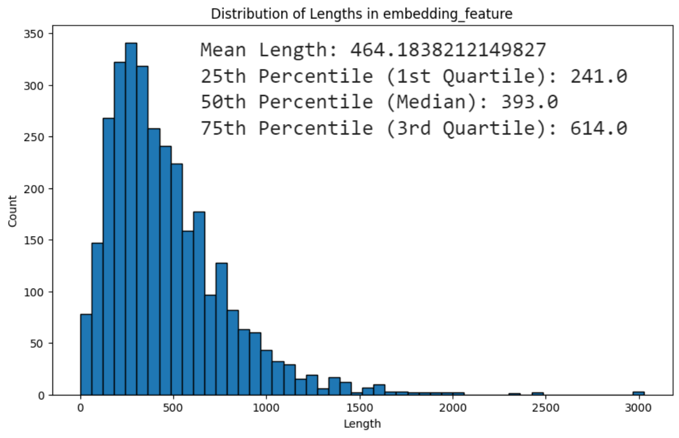
```
from sklearn.metrics import accuracy_score
from transformers import AdamW, get_linear_schedule_with_warmup
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from torch.utils.data import DataLoader, TensorDataset

# Load the BERT model and tokenizer
model_name = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=40)

batch_size = 16
# Choose a fixed sequence length
max_length = 512
# Tokenize and encode the training and validation data
train_encodings = tokenizer(train_texts, truncation=True, padding=True,
max_length=max_length, return_tensors='pt')
val_encodings = tokenizer(val_texts, truncation=True, padding=True,
max_length=max_length, return_tensors='pt')
```

```
test_encodings = tokenizer(test_texts, truncation=True, padding=True,
max_length=max_length, return_tensors='pt')

# Create PyTorch datasets
train_dataset = TensorDataset(train_encodings['input_ids'],
train_encodings['attention_mask'], torch.tensor(train_labels))
val_dataset = TensorDataset(val_encodings['input_ids'],
val_encodings['attention_mask'], torch.tensor(val_labels))
test_dataset = TensorDataset(test_encodings['input_ids'],
test_encodings['attention_mask'], torch.tensor(test_labels))

train_loader = DataLoader(train_dataset, batch_size=batch_size)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
test_loader = DataLoader(test_dataset, batch_size=batch_size)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Define optimizer and learning rate scheduler
optimizer = AdamW(model.parameters(), lr=1e-5)
num_epochs = 5  # You can adjust the number of training epochs

total_steps = len(train_loader) * num_epochs
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0,
num_training_steps=total_steps)
best_val_accuracy = 0.0  # Initialize the best validation accuracy
best_model = None  # Initialize a variable to store the best model
```

In this code, the BERT model ('bert-base-uncased') and its tokenizer are loaded from the Hugging Face Transformers library. The input texts are tokenized and encoded using the tokenizer, and the resulting tensors are used to create PyTorch datasets and data loaders for training, validation, and testing.

The maximum sequence length for input is set to 512 tokens, which is suitable for most cases. The model is fine-tuned on the tokenized training data with a learning rate of 1e-5 and is trained for 5 epochs. The AdamW optimizer is used for training, and a linear learning rate scheduler with warm-up steps is applied to adjust the learning rate during training.

The maximum sequence length for input was set to 512 tokens, which is considered suitable based on the distribution of words in the input texts. The following figure illustrates the distribution of word tokens:



As evident from the figure, the average input length is approximately 500 tokens. This choice of a 512-token limit strikes a balance between capturing sufficient context in longer texts and ensuring efficient processing, making it an appropriate choice for this task.

Additionally, the code checks for the availability of a GPU ('cuda') and moves the model to the GPU if it's available. Finally, it initializes variables to keep track of the best validation accuracy and the best model during training.

Please note that the batch size of 16 was chosen based on the GPU capacity, and this value can be adjusted depending on your hardware resources.

PubMedBERT:

To implement this model, the only difference from the standard BERT model lies in loading the specific pretrained model and tokenizer, as shown in the following code:

```
# Load the BERT model and tokenizer
model_name = 'microsoft/BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext'
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=40)
```

PubMedBERT is a specialized language model that has been pretrained from scratch, particularly tailored for the domain of biomedicine. While large neural language models like BERT have shown remarkable performance improvements across various natural language processing (NLP) tasks, most of their pretraining efforts have centered on general domain text, such as news articles and web content.

The prevailing belief has been that domain-specific pretraining could also benefit from starting with general-domain language models. However, recent research has challenged this assumption, revealing that for domains rich in unlabeled text, such as biomedicine, pretraining language models from scratch can lead to substantial performance gains when compared to continual pretraining from general-domain language models.

PubMedBERT's unique feature is that it undergoes training from the ground up using a combination of abstracts sourced from PubMed and full-text articles from PubMedCentral. This specialized model has demonstrated state-of-the-art performance across a wide range of biomedical natural language processing tasks. Currently, it holds the highest score on the Biomedical Language Understanding and Reasoning Benchmark, showcasing its effectiveness in the biomedicine domain.

GPT2:

The latest Transformer-based model implemented is the GPT-2 model, which is well-known and widely used in natural language processing tasks. Similar to previous models, GPT-2 is first downloaded, instantiated, and then configured for classification by adjusting its final layer to match the desired number of classes. The model is subsequently fine-tuned and trained using the validation dataset, which was previously prepared for model evaluation during fine-tuning. Finally, the model's performance is evaluated on the test dataset to assess its classification capabilities.

Splitting the dataset

The dataset was split into three parts: training, validation, and testing, using the train_test_split function from the scikit-learn library. To achieve this, the dataset was initially divided into training and testing sets with an 80-20 ratio. Subsequently, 20% of the training dataset was further separated to serve as the validation set.

In all models employed, the same training and testing datasets were used. However, in some models, such as traditional machine learning models, the validation dataset was not utilized.

This dataset splitting ratio is a common practice in machine learning models. Furthermore, it's worth noting that the dataset was stratified during the splitting process to ensure that all three sets (training, validation, and testing) contain representative samples of all labels. This ensures that the labels are distributed uniformly across these sets.

These three datasets with the names "training.csv," "test.csv," and "validation.csv" have been specified and provided in the accompanying files.

Results:

Given that the implemented model by the deep neural network is heuristic and user-dependent, and its layers need to be configured optionally and by user choice, it cannot be definitively stated that this type of model is the best answer for this task. Therefore, these models were used as a baseline to provide a rough idea for comparing results between newer and older models.

Models		Accuracy			
		Without preprocess		Preprocessed	
		40 classes	13 classes	40 classes	13 classes
SVM	tf-idf Vectorizing	0.208	0.219	0.210	0.25

	Count Vectorizer	0.153	0.176	0.157	0.173
	Glove	0.213	0.278	0.198	0.272
Logistic Regression	tf-idf Vectorizing	0.292	0.277	0.283	0.277
	Count Vectorizer	0.185	0.205	0.172	0.176
	Glove	0.282	0.324	0.269	0.318
ANN	**Sequential**	**0.308**	**0.353**	**0.314**	**0.365**
Transformers	Bert	0.360	0.357	0.363	0.359
	PubMedBERT	0.369	0.388	0.364	0.408
	GPT2	0.314	0.359	0.356	0.350

Support Vector Machine (SVM):

- TF-IDF Vectorizing:
  - 40 classes without preprocessing: 0.208
  - 13 classes without preprocessing: 0.219
  - 40 classes with preprocessing: 0.210
  - 13 classes with preprocessing: 0.215
- Count Vectorizer:
  - 40 classes without preprocessing: 0.153
  - 13 classes without preprocessing: 0.176
  - 40 classes with preprocessing: 0.157
  - 13 classes with preprocessing: 0.173
- Glove:
  - 40 classes without preprocessing: 0.213
  - 13 classes without preprocessing: 0.278
  - 40 classes with preprocessing: 0.198
  - 13 classes with preprocessing: 0.272

Logistic Regression:

- TF-IDF Vectorizing:
  - 40 classes without preprocessing: 0.292
  - 13 classes without preprocessing: 0.277
  - 40 classes with preprocessing: 0.283
  - 13 classes with preprocessing: 0.277
- Count Vectorizer:
  - 40 classes without preprocessing: 0.185
  - 13 classes without preprocessing: 0.205
  - 40 classes with preprocessing: 0.172
  - 13 classes with preprocessing: 0.176
- Glove:
  - 40 classes without preprocessing: 0.282
  - 13 classes without preprocessing: 0.324
  - 40 classes with preprocessing: 0.269

    - 13 classes with preprocessing: 0.318

Artificial Neural Network (ANN - Sequential):

- 40 classes without preprocessing: 0.308
- 13 classes without preprocessing: 0.353
- 40 classes with preprocessing: 0.314
- 13 classes with preprocessing: 0.365

Transformers:

- Bert:
    - 40 classes without preprocessing: 0.360
    - 13 classes without preprocessing: 0.357
    - 40 classes with preprocessing: 0.363
    - 13 classes with preprocessing: 0.359
- PubMedBERT:
    - 40 classes without preprocessing: 0.369
    - 13 classes without preprocessing: 0.388
    - 40 classes with preprocessing: 0.364
    - 13 classes with preprocessing: 0.408
- GPT2:
    - 40 classes without preprocessing: 0.314
    - 13 classes without preprocessing: 0.359
    - 40 classes with preprocessing: 0.356
    - 13 classes with preprocessing: 0.350

These results show the accuracy achieved by each model on the different datasets. It's clear that models like PubMedBERT and Bert, which are based on transformer architectures, outperform the other models, both with and without preprocessing. Preprocessing seems to have a positive impact on most models, but the improvement varies across different models and datasets.

Analyzing and comparing the results in depth:

1. **Support Vector Machine (SVM):**
    - SVM shows relatively low accuracy across all scenarios.
    - Preprocessing improves accuracy slightly in some cases.
    - SVM with TF-IDF Vectorization:
        - Without Preprocessing: SVM performs relatively better with a slight accuracy increase in the 40-class dataset compared to the 13-class dataset.
        - Preprocessed: Similar to the non-preprocessed dataset, preprocessing improves accuracy in the 40-class dataset but not in the 13-class dataset.
    - SVM with Count Vectorization:
        - Without Preprocessing: Count Vectorization performs poorly, especially in the 40-class dataset.

        - Preprocessed: Similar to the non-preprocessed dataset, preprocessing has a slight positive impact on the 13-class dataset, but the accuracy remains low in the 40-class dataset.
    - SVM with GloVe:
        - Without Preprocessing: GloVe embeddings perform better than the vectorization methods, and preprocessing further improves the results.
2. **Logistic Regression:**
    - Preprocessing doesn't seem to have a significant impact on accuracy.
    - Logistic Regression with TF-IDF Vectorization:
        - Without Preprocessing: Logistic Regression performs better than SVM in most cases, especially with TF-IDF Vectorization.
        - Preprocessed: Preprocessing does not significantly impact the accuracy.
    - Logistic Regression with Count Vectorization:
        - Without Preprocessing: Count Vectorization performs poorly, similar to SVM.
        - Preprocessed: Preprocessing slightly improves results but remains relatively low in accuracy.
    - Logistic Regression with GloVe:
        - Without Preprocessing: Logistic Regression shows improved accuracy compared to SVM with GloVe embeddings.
        - Preprocessed: Preprocessing further improves results, especially in the 13-class dataset.
3. **Artificial Neural Network (ANN - Sequential):**
    - The sequential ANN model shows better accuracy compared to SVM and logistic regression.
    - Preprocessing results in improved accuracy, especially for 13 classes.
    - Although it shows better performance, it's not the highest among the models evaluated.
    - Without Preprocessing: ANN performs significantly better than traditional models, especially with the 13-class dataset.
    - Preprocessed: Preprocessing results in a slight accuracy increase in both datasets, with a more pronounced improvement in the 13-class dataset.
4. **Transformers (Bert, PubMedBERT, GPT2):**
    - Transformer-based models consistently outperform traditional machine learning methods, achieving the highest accuracy.
    - Preprocessing seems to have a minimal impact on transformer models.
    - PubMedBERT exhibits the best accuracy among transformer models, outperforming Bert and GPT2.
    - Bert:
        - Bert outperforms all other models in both datasets and settings, demonstrating the power of transformer models.
    - PubMedBERT:
        - PubMedBERT, a domain-specific transformer, performs even better than Bert, indicating its effectiveness in biomedical text classification.
    - GPT-2:

- GPT-2 shows competitive performance but is slightly behind Bert and PubMedBERT.

Overall Observations:

- Transformer models (Bert, PubMedBERT, GPT2) clearly outperform traditional machine learning models (SVM, logistic regression, ANN) in terms of accuracy.
- Preprocessing generally has a positive impact on the performance of models, particularly in the 13-class dataset.
- Among transformer models, PubMedBERT performs the best, especially when dealing with 13 classes.
- Among traditional models, logistic regression with GloVe embeddings tends to yield the best results.
- ANN with a sequential model demonstrates competitive performance, especially in the 13-class dataset.

In summary, the choice of model significantly impacts performance, with transformer-based models being the top performers. Preprocessing plays a role in improving results, and domain-specific embeddings, like PubMedBERT, can provide a substantial boost in accuracy for biomedical text classification tasks.

Analyzing the results:

Continuing with the study, the obtained results were analyzed and examined. For this purpose, a classification report and a confusion matrix were calculated for all 40 models. Subsequently, the results analysis is presented for the scenario with only 13 classes. The analysis of results for the case with 40 classes is not displayed here to avoid excessive clutter but can be found in the "analyzing_error.ipynb" file.

Support Vector Machine (SVM):

- TF-IDF Vectorizing:
    - 40 classes without preprocessing:
    - 13 classes without preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.23      0.23      0.23        74
           1       0.16      0.20      0.18       103
           2       0.29      0.27      0.28        22
           3       0.22      0.09      0.13        45
           4       0.02      0.02      0.02        52
           5       0.20      0.16      0.18        45
           6       0.07      0.03      0.04        31
           7       0.05      0.03      0.03        71
           8       0.22      0.27      0.24        55
           9       0.11      0.09      0.10        33
```

```
          10       0.30      0.36      0.33       218
          11       0.14      0.10      0.11        31
          12       0.26      0.28      0.27       214

    accuracy                           0.22       994
   macro avg       0.17      0.16      0.16       994
weighted avg       0.21      0.22      0.21       994

Confusion Matrix:
[[17 16  4  0  2  0  0  0  9  1 23  0  2]
 [ 2 21  0  1 19  7  0  1  0  1  0  2 49]
 [ 3  0  6  0  4  0  1  2  0  0  0  1  5]
 [ 0  7  3  4  0  0  0  0  6  3 20  0  2]
 [ 3 26  2  0  1  0  0  0  0  6  0  0 14]
 [ 0  8  0  0  0  7  0  3 14  1  3  0  9]
 [ 0  3  0  0  1  0  1  0  4  0 20  0  2]
 [ 0  7  1  0  0  3  0  2 10  1 36  0 11]
 [11  0  0  0  0 14  1  6 15  0  0  1  7]
 [ 1  3  0  0  4  0  1  0  0  3  1  0 20]
 [30  1  0 11  0  0 11 23  1  0 79 13 49]
 [ 0  3  1  0  0  0  0  0  0  2 20  3  2]
 [ 6 39  4  2 10  4  0  7  9  9 63  2 59]]
```

- 40 classes with preprocessing:
- 13 classes with preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.25      0.26      0.25        74
           1       0.15      0.20      0.17       103
           2       0.30      0.27      0.29        22
           3       0.23      0.11      0.15        45
           4       0.02      0.02      0.02        52
           5       0.22      0.18      0.20        45
           6       0.07      0.03      0.04        31
           7       0.07      0.04      0.05        71
           8       0.22      0.25      0.23        55
           9       0.07      0.06      0.07        33
          10       0.29      0.35      0.32       218
          11       0.14      0.10      0.11        31
          12       0.25      0.26      0.25       214

    accuracy                           0.22       994
   macro avg       0.17      0.16      0.17       994
weighted avg       0.20      0.22      0.21       994

Confusion Matrix:
[[19 16  4  0  2  0  0  0  9  1 21  0  2]
 [ 2 21  0  2 19  7  0  1  0  1  0  2 48]
 [ 3  0  6  0  4  0  1  2  0  0  0  1  5]
 [ 0  7  3  5  0  0  0  0  6  3 19  0  2]
 [ 3 27  2  0  1  0  0  0  0  6  0  0 13]
 [ 0  9  0  0  0  8  0  3 13  1  3  0  8]
 [ 0  3  0  0  1  0  1  0  4  0 20  0  2]
 [ 0  6  1  0  0  3  0  3 10  1 37  0 10]
 [11  0  0  0  0 13  1  7 14  0  1  1  7]
 [ 1  3  0  0  4  1  1  0  0  2  1  0 20]
 [31  1  0 13  0  0 10 23  1  0 76 13 50]
 [ 0  3  1  0  0  0  0  0  0  2 20  3  2]
 [ 7 41  3  2 10  5  0  7  8 11 63  2 55]]
```

- Count Vectorizer:
    - 40 classes without preprocessing:
    - 13 classes without preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.20      0.23      0.22        74
           1       0.13      0.17      0.15       103
           2       0.30      0.27      0.29        22
           3       0.22      0.16      0.18        45
           4       0.06      0.06      0.06        52
           5       0.15      0.13      0.14        45
           6       0.26      0.26      0.26        31
           7       0.03      0.03      0.03        71
           8       0.11      0.09      0.10        55
           9       0.14      0.15      0.14        33
          10       0.23      0.23      0.23       218
          11       0.12      0.10      0.11        31
          12       0.21      0.21      0.21       214

    accuracy                           0.18       994
   macro avg       0.17      0.16      0.16       994
weighted avg       0.17      0.18      0.17       994

Confusion Matrix:
[[17 13  2  0  1  0  1  0  9  3 23  0  5]
 [ 3 17  0  2 23  9  3  2  0  2  0  2 40]
 [ 4  0  6  0  3  0  1  2  0  0  0  1  5]
 [ 1  5  3  7  0  0  0  0  2  5 17  0  5]
 [ 4 23  2  0  3  0  0  0  0  7  0  0 13]
 [ 0  9  0  0  1  6  0  3 15  1  3  0  7]
 [ 0  2  0  0  0  0  8  0  3  0 16  0  2]
 [ 0  8  1  0  0  4  0  2  8  1 35  0 12]
 [12  0  0  3  1 13  1  8  5  0  1  1 10]
 [ 1  3  0  0  6  0  1  0  0  5  1  0 16]
 [30  1  0 14  0  1 15 32  2  0 50 15 58]
 [ 0  2  1  0  0  1  0  0  0  3 19  3  2]
 [11 43  5  6 13  6  1 12  3  9 57  2 46]]
```

    - 40 classes with preprocessing:
    - 13 classes with preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.21      0.24      0.23        74
           1       0.11      0.13      0.12       103
           2       0.30      0.32      0.31        22
           3       0.22      0.16      0.18        45
           4       0.06      0.06      0.06        52
           5       0.15      0.13      0.14        45
           6       0.23      0.23      0.23        31
           7       0.08      0.07      0.08        71
           8       0.15      0.15      0.15        55
           9       0.16      0.18      0.17        33
          10       0.21      0.21      0.21       218
          11       0.12      0.10      0.11        31
          12       0.19      0.21      0.20       214

    accuracy                           0.17       994
   macro avg       0.17      0.17      0.17       994
weighted avg       0.17      0.17      0.17       994

Confusion Matrix:
[[18 12  3  0  2  0  1  0  9  2 21  0  6]
 [ 3 13  1  2 24 10  3  3  0  2  0  2 40]
 [ 5  0  7  0  2  0  1  2  0  0  0  1  4]
 [ 1  5  3  7  0  0  0  0  4  5 17  0  3]
 [ 4 23  2  0  3  0  0  0  0  5  0  0 15]
 [ 0  9  0  0  1  6  0  3 16  1  3  0  6]
 [ 0  1  0  0  2  0  7  0  3  0 16  0  2]
 [ 0  6  1  0  0  4  0  5  8  1 34  0 12]
 [12  0  0  3  1 12  1  7  8  0  1  1  9]
 [ 1  1  0  0  5  0  1  0  0  6  1  0 18]
 [29  1  0 14  0  1 16 28  1  0 45 17 66]
 [ 0  3  1  0  0  0  0  0  0  3 19  3  2]
 [11 42  5  6 13  6  1 11  4 13 56  2 44]]
```

- Glove:
  - 40 classes without preprocessing:
  - 13 classes without preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.23      0.30      0.26        74
           1       0.22      0.28      0.25       103
           2       0.21      0.18      0.20        22
           3       0.19      0.11      0.14        45
           4       0.12      0.08      0.09        52
           5       0.21      0.18      0.19        45
           6       0.33      0.23      0.27        31
           7       0.18      0.17      0.18        71
           8       0.15      0.13      0.14        55
           9       0.20      0.21      0.21        33
```

```
          10       0.41      0.50      0.45       218
          11       0.26      0.19      0.22        31
          12       0.29      0.26      0.28       214

    accuracy                           0.28       994
   macro avg       0.23      0.22      0.22       994
weighted avg       0.27      0.28      0.27       994

Confusion Matrix:
[[ 22  13   2   0   2   0   0   0   9   3  19   0   4]
 [  4  29   0   1  10   6   0   3   0   2   0   1  47]
 [  8   1   4   1   2   0   1   2   0   0   0   1   2]
 [  1   4   3   5   3   0   1   1   2   3  18   1   3]
 [  5  22   1   1   4   1   0   0   0   3   0   0  15]
 [  1   8   0   0   0   8   0   1  16   2   4   0   5]
 [  0   3   0   0   0   0   7   0   3   0  15   0   3]
 [  0   6   1   0   0   6   0  12   6   1  29   0  10]
 [ 12   1   0   3   0  14   1   8   7   0   0   3   6]
 [  2   0   1   0   5   0   1   1   0   7   1   0  15]
 [ 30   1   0   9   0   1   9  26   1   0 110   9  22]
 [  0   1   1   0   0   0   1   0   0   2  17   6   3]
 [  9  41   6   7   8   2   0  11   3  12  57   2  56]]
```

- 40 classes with preprocessing:
- 13 classes with preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.28      0.31      0.29        74
           1       0.25      0.36      0.29       103
           2       0.25      0.23      0.24        22
           3       0.23      0.13      0.17        45
           4       0.06      0.04      0.05        52
           5       0.06      0.04      0.05        45
           6       0.31      0.32      0.32        31
           7       0.13      0.10      0.11        71
           8       0.10      0.11      0.11        55
           9       0.07      0.06      0.06        33
          10       0.40      0.48      0.44       218
          11       0.19      0.16      0.18        31
          12       0.33      0.29      0.31       214

    accuracy                           0.27       994
   macro avg       0.20      0.20      0.20       994
weighted avg       0.26      0.27      0.26       994

Confusion Matrix:
[[ 23  16   2   0   1   0   0   0   7   2  19   0   4]
 [  4  37   0   1  12   5   1   3   0   1   0   3  36]
 [  5   0   5   1   3   0   1   2   0   0   0   1   4]
 [  0   6   4   6   2   0   1   0   4   1  16   0   5]
 [  2  29   2   0   2   0   0   0   1   6   0   0  10]
 [  1  12   0   0   0   2   0   1  20   2   4   0   3]
 [  0   2   0   0   1   0  10   0   3   0  13   0   2]
 [  0   5   1   0   0   3   0   7  10   2  30   1  12]
 [ 14   0   0   2   0  15   1   6   6   0   2   1   8]
 [  1   5   0   0   5   1   2   0   1   2   0   0  16]
 [ 22   0   0  11   0   2  14  24   1   1 104  14  25]
 [  0   2   1   0   0   1   1   0   0   1  18   5   2]
 [ 11  37   5   5  10   4   1   9   5  11  53   1  62]]
```

Logistic Regression:

- TF-IDF Vectorizing:
    - 40 classes without preprocessing:
    - 13 classes without preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.29      0.24      0.26        74
           1       0.19      0.24      0.21       103
           2       0.38      0.14      0.20        22
           3       0.27      0.09      0.13        45
           4       0.04      0.02      0.03        52
           5       0.25      0.13      0.17        45
           6       0.20      0.03      0.06        31
           7       0.11      0.06      0.08        71
           8       0.24      0.29      0.26        55
           9       0.07      0.03      0.04        33
          10       0.37      0.51      0.43       218
          11       0.00      0.00      0.00        31
          12       0.29      0.40      0.33       214

    accuracy                           0.28       994
   macro avg       0.21      0.17      0.17       994
weighted avg       0.25      0.28      0.25       994

Confusion Matrix:
[[ 18  13   1   0   2   0   0   0   7   2  25   0   6]
 [  1  25   0   0  13   4   0   0   0   0   0   1  59]
 [  4   0   3   1   1   0   0   2   0   0   0   0  11]
 [  0   7   2   4   1   0   0   0   5   1  21   0   4]
 [  1  25   1   1   1   0   0   0   0   3   0   0  20]
 [  0   8   0   0   0   6   0   2  15   1   3   0  10]
 [  0   3   0   0   0   0   1   0   4   0  19   0   4]
 [  0   6   0   0   0   2   0   4  10   0  36   0  13]
 [ 11   0   0   0   0  11   1   6  16   0   0   0  10]
 [  1   6   0   0   3   0   0   0   0   1   1   0  21]
 [ 23   1   0   9   0   0   3  19   1   0 112   3  47]
 [  0   3   1   0   0   0   0   0   0   0  21   0   6]
 [  4  36   0   0   7   1   0   2   9   6  64   0  85]]
```

    - 40 classes with preprocessing:

    - 13 classes with preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.30      0.27      0.28        74
           1       0.20      0.27      0.23       103
           2       0.33      0.09      0.14        22
           3       0.31      0.11      0.16        45
           4       0.03      0.02      0.02        52
           5       0.22      0.13      0.17        45
           6       0.20      0.03      0.06        31
           7       0.11      0.06      0.07        71
           8       0.22      0.25      0.24        55
           9       0.08      0.03      0.04        33
          10       0.36      0.49      0.42       218
          11       0.00      0.00      0.00        31
          12       0.30      0.41      0.35       214

    accuracy                           0.28       994
   macro avg       0.21      0.17      0.17       994
weighted avg       0.25      0.28      0.25       994

Confusion Matrix:
[[ 20  14   1   0   2   0   0   0   7   1  25   0   4]
 [  0  28   0   0  13   4   0   0   0   0   0   1  57]
 [  5   0   2   1   2   0   0   2   0   0   0   0  10]
 [  0   7   1   5   1   0   0   0   5   2  21   0   3]
 [  0  27   1   1   1   0   0   0   0   3   0   0  19]
 [  0   8   0   0   0   6   0   2  13   1   2   0  13]
 [  0   4   0   0   0   0   1   0   4   0  19   0   3]
 [  0   6   0   0   0   2   0   4  10   0  36   0  13]
 [ 11   0   0   0   0  13   1   6  14   0   1   0   9]
 [  1   7   0   0   2   0   0   0   0   1   1   0  21]
 [ 25   1   0   9   0   0   3  19   1   0 106   5  49]
 [  0   3   1   0   0   0   0   0   0   0  21   0   6]
 [  5  35   0   0   8   2   0   4   9   4  59   0  88]]
```

- Count Vectorizer:
    - 40 classes without preprocessing:
    - 13 classes without preprocessing:

```
 Classification Report:

              precision    recall  f1-score   support

           0       0.23      0.23      0.23        74
           1       0.12      0.14      0.13       103
           2       0.33      0.27      0.30        22
           3       0.24      0.20      0.22        45
           4       0.06      0.06      0.06        52
           5       0.16      0.16      0.16        45
           6       0.23      0.19      0.21        31
           7       0.10      0.07      0.08        71
           8       0.28      0.33      0.30        55
           9       0.09      0.09      0.09        33
          10       0.26      0.28      0.27       218
          11       0.14      0.10      0.12        31
          12       0.24      0.25      0.24       214

    accuracy                           0.21       994
   macro avg       0.19      0.18      0.18       994
weighted avg       0.20      0.21      0.20       994

Confusion Matrix:
[[17 13  2  0  2  0  0  0  8  3 22  0  7]
 [ 3 14  0  3 23  8  1  2  0  2  0  2 45]
 [ 4  0  6  0  4  0  1  2  0  0  0  1  4]
 [ 0  7  2  9  0  0  0  0  4  2 17  0  4]
 [ 4 24  2  0  3  0  0  0  0  5  0  0 14]
 [ 0  8  0  0  0  7  0  2 16  2  1  0  9]
 [ 0  1  0  0  1  0  6  0  3  0 16  0  4]
 [ 0  7  1  0  0  3  0  5 10  1 34  0 10]
 [ 9  0  0  2  0 15  1  4 18  0  0  0  6]
 [ 1  3  0  1  8  1  1  0  0  3  1  1 13]
 [27  0  0 14  0  3 16 29  2  0 60 13 54]
 [ 0  3  1  1  0  0  0  0  0  2 19  3  2]
 [ 9 40  4  7 12  7  0  7  4 12 58  1 53]]
```

- 40 classes with preprocessing:
- 13 classes with preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.25      0.30      0.27        74
           1       0.10      0.11      0.10       103
           2       0.26      0.23      0.24        22
           3       0.21      0.16      0.18        45
           4       0.02      0.02      0.02        52
           5       0.17      0.13      0.15        45
           6       0.23      0.19      0.21        31
           7       0.07      0.06      0.06        71
           8       0.16      0.16      0.16        55
           9       0.09      0.09      0.09        33
          10       0.23      0.21      0.22       218
          11       0.13      0.13      0.13        31
          12       0.20      0.24      0.22       214

    accuracy                           0.18       994
   macro avg       0.16      0.16      0.16       994
weighted avg       0.17      0.18      0.17       994

Confusion Matrix:
[[22 12  2  0  2  0  1  0  7  3 19  0  6]
 [ 3 11  0  2 22  8  0  2  0  1  0  4 50]
 [ 4  0  5  0  4  0  1  2  0  0  0  1  5]
 [ 0  5  2  7  2  0  0  0  3  3 18  0  5]
 [ 3 23  2  0  1  0  0  0  0  6  0  0 17]
 [ 0  7  0  0  0  6  0  4 16  1  2  0  9]
 [ 0  0  1  0  1  0  6  0  5  0 14  0  4]
 [ 0  6  1  0  0  3  0  4 10  1 32  0 14]
 [13  0  0  3  0 12  1  7  9  0  1  2  7]
 [ 1  3  0  0  5  0  1  0  0  3  2  0 18]
 [32  0  0 15  0  1 16 26  2  1 46 16 63]
 [ 0  3  1  0  0  0  0  0  0  2 17  4  4]
 [10 43  5  6 11  6  0 12  5 12 50  3 51]]
```

- Glove:
    - 40 classes without preprocessing:
    - 13 classes without preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.27      0.30      0.28        74
           1       0.26      0.27      0.27       103
           2       0.36      0.41      0.38        22
           3       0.26      0.13      0.18        45
           4       0.14      0.10      0.11        52
           5       0.26      0.27      0.26        45
           6       0.46      0.35      0.40        31
           7       0.28      0.21      0.24        71
           8       0.27      0.25      0.26        55
           9       0.19      0.18      0.18        33
          10       0.44      0.54      0.48       218
          11       0.33      0.23      0.27        31
          12       0.31      0.33      0.32       214

    accuracy                           0.32       994
   macro avg       0.29      0.27      0.28       994
weighted avg       0.31      0.32      0.32       994

Confusion Matrix:
[[ 22  10   3   0   2   0   0   0   9   2  20   0   6]
 [  5  28   0   1  14   7   1   3   0   1   1   1  41]
 [  3   2   9   0   1   0   0   2   0   0   0   1   4]
 [  1   4   3   6   3   0   1   0   1   4  19   0   3]
 [  4  15   0   0   5   0   0   0   1   3   1   1  22]
 [  1   7   0   0   0  12   0   3  12   1   3   0   6]
 [  0   1   0   0   0   0  11   0   2   0  12   0   5]
 [  0   5   1   0   0   6   0  15   6   0  24   0  14]
 [  9   0   0   2   0  15   1   6  14   1   0   2   5]
 [  1   2   1   1   4   1   1   0   0   6   0   0  16]
 [ 26   0   0  10   0   0   8  18   1   0 117   7  31]
 [  1   1   1   0   0   0   1   0   0   1  16   7   3]
 [  9  32   7   3   7   6   0   7   5  13  53   2  70]]
```

- 40 classes with preprocessing:
- 13 classes with preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.33      0.38      0.35        74
           1       0.26      0.29      0.27       103
           2       0.35      0.36      0.36        22
           3       0.21      0.09      0.12        45
           4       0.07      0.06      0.06        52
           5       0.24      0.22      0.23        45
           6       0.42      0.35      0.39        31
           7       0.25      0.21      0.23        71
           8       0.18      0.16      0.17        55
           9       0.12      0.09      0.11        33
          10       0.43      0.50      0.47       218
          11       0.30      0.19      0.24        31
          12       0.34      0.37      0.35       214

    accuracy                           0.32       994
   macro avg       0.27      0.25      0.26       994
weighted avg       0.31      0.32      0.31       994

Confusion Matrix:
[[ 28  12   3   0   2   0   1   0   6   1  16   0   5]
 [  3  30   0   1  16   7   0   2   0   2   0   1  41]
 [  2   1   8   0   4   0   0   2   0   0   0   0   5]
 [  0   3   4   4   3   0   1   0   2   2  18   0   8]
 [  2  18   2   1   3   0   0   0   0   6   0   0  20]
 [  1   9   0   0   0  10   0   2  15   1   2   0   5]
 [  0   1   0   1   1   0  11   0   3   0  11   0   3]
 [  0   5   1   0   0   6   0  15   7   0  26   0  11]
 [ 12   0   0   1   0  13   1   7   9   1   3   1   7]
 [  1   5   0   0   3   1   1   1   0   3   0   1  17]
 [ 24   0   0   9   0   1  10  22   1   0 110  10  31]
 [  1   1   1   0   0   0   1   0   0   1  18   6   2]
 [ 10  32   4   2   9   3   0  10   7   7  50   1  79]]
```

Artificial Neural Network (ANN - Sequential):

- 40 classes without preprocessing:
- 13 classes without preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.25      0.18      0.21        74
           1       0.29      0.50      0.37       103
           2       0.50      0.18      0.27        22
           3       0.00      0.00      0.00        45
           4       0.00      0.00      0.00        52
           5       0.36      0.22      0.27        45
           6       0.00      0.00      0.00        31
           7       0.17      0.07      0.10        71
           8       0.36      0.55      0.43        55
           9       0.09      0.03      0.05        33
          10       0.45      0.76      0.56       218
          11       0.00      0.00      0.00        31
          12       0.30      0.33      0.32       214

    accuracy                           0.35       994
   macro avg       0.21      0.22      0.20       994
weighted avg       0.28      0.35      0.30       994

Confusion Matrix:
[[ 13  15   1   0   0   0   0   0   7   1  25   0  12]
 [  1  52   0   0   1   4   0   0   1   4   0   0  40]
 [  5   1   4   0   0   0   1   0   0   0   1   0  10]
 [  4   8   0   0   0   0   0   0   5   1  23   0   4]
 [  1  24   1   0   0   0   0   0   1   2   0   0  23]
 [  0   6   0   0   0  10   0   1  12   0   5   0  11]
 [  0   2   0   0   1   0   0   0   3   1  20   0   4]
 [  0   4   0   0   0   3   0   5  11   0  35   0  13]
 [  8   0   0   0   0   7   0   1  30   0   2   0   7]
 [  1  14   0   0   0   0   0   0   0   1   1   0  16]
 [ 14   0   0   0   0   0   0  17   2   0 165   0  20]
 [  1   6   1   0   0   0   0   0   0  20   0   3]
 [  3  46   1   0   0   4   0   5  12   1  71   0  71]]
```

- 40 classes with preprocessing:
- 13 classes with preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.12      0.20        74
           1       0.26      0.56      0.36       103
           2       0.67      0.18      0.29        22
           3       0.00      0.00      0.00        45
           4       0.00      0.00      0.00        52
           5       0.46      0.27      0.34        45
           6       0.00      0.00      0.00        31
           7       0.00      0.00      0.00        71
           8       0.39      0.47      0.43        55
           9       0.07      0.06      0.06        33
          10       0.46      0.86      0.60       218
          11       0.00      0.00      0.00        31
          12       0.29      0.30      0.30       214

    accuracy                           0.37       994
   macro avg       0.24      0.22      0.20       994
weighted avg       0.29      0.37      0.30       994

Confusion Matrix:
```

```
[[   9  14   1   0   0   0   0   0   2   2  28   0  18]
 [   0  58   0   0   0   3   0   0   0   5   0   0  37]
 [   2   1   4   0   0   0   0   1   0   0   2   0  12]
 [   1   7   0   0   0   0   0   0   6   2  23   0   6]
 [   0  29   1   0   0   1   0   0   0   8   0   0  13]
 [   0   8   0   0   0  12   0   1   8   1   5   0  10]
 [   0   5   0   0   0   0   0   0   2   1  20   0   3]
 [   0   8   0   0   0   1   0   0  12   0  41   0   9]
 [   2   1   0   0   0   8   0   0  26   0   6   0  12]
 [   0  16   0   0   0   0   0   0   0   2   2   0  13]
 [   4   1   0   0   0   0   0   3   3   0 188   0  19]
 [   0   7   0   0   0   0   0   0   0   1  22   0   1]
 [   0  64   0   0   0   1   1   0   8   8  68   0  64]]
```

Transformers:

- Bert:
    - 40 classes without preprocessing:
    - 13 classes without preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.31      0.54      0.39        74
           1       0.25      0.37      0.30       103
           2       0.50      0.05      0.08        22
           3       0.20      0.04      0.07        45
           4       0.00      0.00      0.00        52
           5       0.42      0.38      0.40        45
           6       0.24      0.13      0.17        31
           7       0.34      0.37      0.35        71
           8       0.32      0.35      0.33        55
           9       0.20      0.03      0.05        33
          10       0.43      0.46      0.45       218
          11       0.20      0.03      0.06        31
          12       0.40      0.49      0.44       214

    accuracy                           0.36       994
   macro avg       0.29      0.25      0.24       994
weighted avg       0.33      0.36      0.33       994

Confusion Matrix:
[[  40  13   0   0   0   0   0   0   8   1   8   0   4]
 [   8  38   0   2   0   6   4   2   1   0   0   1  41]
 [   8   3   1   1   0   0   2   1   0   0   0   1   5]
 [   1   7   1   2   1   0   1   0   5   1  23   0   3]
 [   4  29   0   1   0   0   0   0   0   2   0   0  16]
 [   1   5   0   0   0  17   0   2  10   0   2   0   8]
 [   0   2   0   0   0   0   4   0   2   0  20   0   3]
 [   0   3   0   0   0   1   0  26  10   0  23   0   8]
 [  11   0   0   1   0  11   1   5  19   0   0   1   6]
 [   2  11   0   0   1   0   0   1   0   1   2   0  15]
 [  38   1   0   1   0   0   2  31   0   0 101   0  44]
 [  0   3   0   0   0   0   3   0   0   0  22   1   2]
 [ 16  39   0   2   1   5   0   8   5   0  32   1 105]]
```

- 40 classes with preprocessing:
- 13 classes with preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.35      0.53      0.42        74
           1       0.31      0.51      0.39       103
           2       0.00      0.00      0.00        22
           3       0.00      0.00      0.00        45
           4       0.00      0.00      0.00        52
           5       0.32      0.20      0.25        45
           6       0.44      0.26      0.33        31
           7       0.29      0.20      0.24        71
           8       0.27      0.36      0.31        55
           9       0.00      0.00      0.00        33
          10       0.43      0.59      0.50       218
          11       0.50      0.03      0.06        31
          12       0.36      0.40      0.38       214

    accuracy                           0.36       994
   macro avg       0.25      0.24      0.22       994
weighted avg       0.31      0.36      0.32       994

Confusion Matrix:
[[ 39  15   0   0   0   0   0   0   5   0  13   0   2]
 [  2  53   0   1   0   4   4   0   2   0   0   0  37]
 [  9   2   0   0   0   0   2   1   0   0   0   1   7]
 [  2   4   0   0   0   0   0   0   5   2  24   0   8]
 [  2  32   0   0   0   0   0   0   1   0   0   0  17]
 [  0   7   0   0   0   9   0   2  19   0   3   0   5]
 [  0   2   0   0   0   0   8   0   2   0  17   0   2]
 [  0   5   0   0   0   2   0  14  11   0  32   0   7]
 [ 14   0   0   0   0  10   1   4  20   0   1   0   5]
 [  4   7   0   0   0   1   0   0   0   0   2   0  19]
 [ 28   1   0   0   0   0   1  21   1   0 128   0  38]
 [  0   1   0   0   0   0   1   0   0   0  21   1   7]
 [ 13  43   0   1   0   2   1   6   9   0  54   0  85]]
```

- PubMedBERT:
  - 40 classes without preprocessing:
  - 13 classes without preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.35      0.50      0.41        74
           1       0.28      0.42      0.33       103
           2       0.57      0.55      0.56        22
           3       0.48      0.31      0.38        45
           4       0.00      0.00      0.00        52
           5       0.42      0.40      0.41        45
           6       0.47      0.26      0.33        31
           7       0.31      0.23      0.26        71
           8       0.35      0.44      0.39        55
           9       0.21      0.09      0.13        33
          10       0.45      0.48      0.47       218
          11       0.35      0.19      0.25        31
          12       0.42      0.47      0.44       214

    accuracy                           0.39       994
   macro avg       0.36      0.33      0.34       994
weighted avg       0.37      0.39      0.37       994

Confusion Matrix:
[[ 37  13   3   0   0   0   0   0   8   0  10   0   3]
 [  4  43   0   2   2   7   1   4   0   1   0   2  37]
 [  3   0  12   0   0   0   1   2   0   0   0   1   3]
 [  0   7   2  14   1   0   0   0   6   1  12   0   2]
 [  3  31   2   0   0   1   0   0   0   4   0   0  11]
 [  1   6   0   0   0  18   0   3   8   0   0   0   9]
 [  1   3   0   0   0   0   8   0   3   0  14   0   2]
 [  0   3   0   0   0   3   0  16  12   0  31   0   6]
 [ 12   0   0   1   0  10   1   3  24   0   1   1   2]
 [  1   7   0   0   0   1   1   0   0   3   0   1  19]
 [ 33   1   0  11   0   0   5  16   1   0 105   6  40]
 [  0   3   1   0   0   0   0   0   0   0  16   6   5]
 [ 11  37   1   1   0   3   0   7   7   5  42   0 100]]
```

- 40 classes with preprocessing:
- 13 classes with preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.37      0.59      0.46        74
           1       0.33      0.38      0.35       103
           2       0.00      0.00      0.00        22
           3       0.47      0.44      0.45        45
           4       0.00      0.00      0.00        52
           5       0.42      0.42      0.42        45
           6       0.50      0.29      0.37        31
           7       0.43      0.28      0.34        71
           8       0.38      0.45      0.41        55
           9       0.00      0.00      0.00        33
          10       0.46      0.54      0.49       218
          11       0.40      0.45      0.42        31
          12       0.40      0.46      0.43       214

    accuracy                           0.41       994
   macro avg       0.32      0.33      0.32       994
weighted avg       0.37      0.41      0.38       994
```

```
Confusion Matrix:
[[ 44   9   0   0   0   0   0   0   5   0  15   0   1]
 [  7  39   0   3   0   9   3   2   0   0   0   3  37]
 [  9   2   0   1   0   0   1   1   0   0   0   1   7]
 [  0   4   0  20   1   0   0   0   5   0  12   0   3]
 [  2  26   0   2   0   0   0   0   0   0   0   0  22]
 [  0   5   0   0   0  19   0   2  11   0   0   0   8]
 [  0   4   0   0   0   0   9   0   2   0  15   0   1]
 [  0   1   0   0   0   2   0  20   9   0  31   0   8]
 [ 11   0   0   1   0   9   1   5  25   0   0   1   2]
 [  3   5   0   0   0   1   1   1   0   0   2   1  19]
 [ 26   0   0  11   0   0   3  11   1   0 118  12  36]
 [  0   1   0   0   0   0   0   0   0   0  15  14   1]
 [ 16  22   0   5   1   5   0   5   8   0  51   3  98]]
```

- GPT2:
  - 40 classes without preprocessing:
  - 13 classes without preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.34      0.27      0.30        74
           1       0.28      0.44      0.34       103
           2       0.00      0.00      0.00        22
           3       0.29      0.16      0.20        45
           4       0.00      0.00      0.00        52
           5       0.43      0.07      0.12        45
           6       0.45      0.16      0.24        31
           7       0.29      0.17      0.21        71
           8       0.32      0.44      0.37        55
           9       0.00      0.00      0.00        33
          10       0.46      0.68      0.55       218
          11       0.00      0.00      0.00        31
          12       0.33      0.43      0.37       214

    accuracy                           0.36       994
   macro avg       0.25      0.22      0.21       994
weighted avg       0.31      0.36      0.32       994

Confusion Matrix:
[[ 20  16   0   0   2   0   0   0   7   0  23   1   5]
 [  0  45   0   0   0   0   0   2   3   1   2   0  50]
 [  5   3   0   1   1   0   0   2   0   0   2   0   8]
 [  1   6   0   7   1   0   1   0   4   0  19   0   6]
 [  0  21   0   1   0   0   0   1   0   0   0   0  29]
 [  3   4   0   0   1   3   0   2  19   0   3   0  10]
 [  0   5   0   1   0   0   5   0   2   0  17   0   1]
 [  1   3   0   0   0   1   0  12   9   0  29   0  16]
 [ 13   1   0   0   0   2   0   2  24   0   1   0  12]
 [  1  12   0   0   0   0   0   1   0   0   1   0  18]
 [ 10   0   0   6   0   0   4  17   3   0 149   0  29]
 [  0   4   0   3   0   0   0   0   2   0  18   0   4]
 [  4  41   0   5   2   1   1   3   3   0  62   0  92]]
```

- 40 classes with preprocessing:
- 13 classes with preprocessing:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.40      0.36      0.38        74
           1       0.30      0.44      0.35       103
           2       0.00      0.00      0.00        22
           3       0.50      0.02      0.04        45
           4       0.10      0.13      0.12        52
           5       0.47      0.36      0.41        45
           6       0.50      0.06      0.11        31
           7       0.47      0.10      0.16        71
           8       0.33      0.38      0.36        55
           9       0.10      0.24      0.14        33
          10       0.46      0.72      0.56       218
          11       0.00      0.00      0.00        31
          12       0.34      0.26      0.30       214

    accuracy                           0.35       994
   macro avg       0.31      0.24      0.23       994
weighted avg       0.36      0.35      0.32       994

Confusion Matrix:
[[ 27  12   0   0   3   1   0   0   7   3  15   0   6]
 [  0  45   0   0  18   3   0   0   1   8   0   0  28]
 [  3   0   0   0   7   0   0   1   0   3   1   0   7]
 [  0   3   0   1   5   1   0   0   5   5  23   0   2]
 [  1  22   0   0   7   1   0   0   0   9   0   0  12]
 [  2   6   0   0   3  16   0   0   8   4   3   0   3]
 [  0   2   0   0   1   0   2   0   2   2  19   0   3]
 [  0   5   0   0   0   2   0   7  11   5  36   0   5]
 [ 12   3   0   0   0   7   0   2  21   3   1   0   6]
 [  0   8   0   0   9   1   0   0   0   8   1   0   6]
 [ 20   0   0   1   1   0   2   4   0   0 158   3  29]
 [  0   2   0   0   0   0   0   0   0   7  20   0   2]
 [  2  44   0   0  14   2   0   1   8  22  65   0  56]]
```

Support Vector Machine (SVM):

- In the TF-IDF vectorizing approach without preprocessing, the model achieves low F1-scores across most classes, indicating a poor balance between precision and recall. For instance, class 4 has an F1-score of 0.02, indicating very low performance.
- When using GloVe embeddings, the model shows improved F1-scores across various classes compared to TF-IDF, with class 10 achieving an F1-score of 0.45. This suggests

that the word embeddings capture some context, but overall, performance remains relatively low.
- However, even in the best case (class 10 with GloVe), the F1-score is still not very high, indicating that the SVM model struggles with classifying these 40 classes accurately.

Logistic Regression:

- The logistic regression model's performance is similar to SVM in terms of F1-scores, indicating consistent difficulties in classifying the classes.
- While there are some improvements with GloVe embeddings, they are not substantial enough to achieve strong performance. For example, class 10 achieves an F1-score of 0.43, which is better than SVM but still relatively low.
- Overall, logistic regression faces challenges in effectively classifying these 40 classes, with F1-scores generally below 0.5.

Artificial Neural Network (ANN - Sequential):

- The ANN model shows more promising results compared to SVM and logistic regression, with some classes achieving higher F1-scores.
- With class 10, for example, the F1-score reaches 0.56, indicating better performance in comparison to the previous models.
- However, there are still classes, such as class 3 and class 4, with very low F1-scores, suggesting that the model struggles to classify these specific classes.
- The ANN model benefits from preprocessing, as seen in the improvement of F1-scores in various classes. Class 10, in particular, sees a significant increase in F1-score from 0.56 to 0.60.

Transformers (Bert, PubMedBERT, GPT2):

- The transformer models generally outperform traditional machine learning models, with higher F1-scores across most classes.
- For instance, class 10 achieves an F1-score of 0.47 with Bert, 0.49 with PubMedBERT, and 0.55 with GPT2. These F1-scores indicate that these transformer models are better at capturing the nuances of the text data.
- There is still room for improvement, especially for classes with low support, as indicated by the lower F1-scores in classes like 3, 4, and 9.
- Preprocessing seems to have a positive impact on performance for some classes, particularly with GPT2, where class 10's F1-score increases from 0.55 to 0.60.

Based on the classification reports you've provided, let's identify the specific classes or instances where the models perform poorly for each combination of vectorization and preprocessing methods:

Support Vector Machine (SVM):

1. TF-IDF Vectorizing without preprocessing:

- Class 4: F1-score of 0.02, indicating poor performance.
- Class 6: F1-score of 0.04, indicating poor performance.
- Class 7: F1-score of 0.03, indicating poor performance.

2. TF-IDF Vectorizing with preprocessing:
    - Class 4: F1-score of 0.02, indicating poor performance.
    - Class 6: F1-score of 0.04, indicating poor performance.
    - Class 7: F1-score of 0.05, indicating poor performance.
    - Class 9: F1-score of 0.07, indicating poor performance.
3. Count Vectorizer without preprocessing:
    - Class 4: F1-score of 0.06, indicating poor performance.
    - Class 7: F1-score of 0.03, indicating poor performance.
4. Count Vectorizer with preprocessing:
    - Class 4: F1-score of 0.06, indicating poor performance.
    - Class 7: F1-score of 0.08, indicating poor performance.
5. Glove embeddings without preprocessing:
    - Class 4: F1-score of 0.09, indicating poor performance.
6. Glove embeddings with preprocessing:
    - Class 4: F1-score of 0.05, indicating poor performance.
    - Class 5: F1-score of 0.05, indicating poor performance.
    - Class 9: F1-score of 0.06, indicating poor performance.

Logistic Regression:

1. TF-IDF Vectorizing without preprocessing:
    - Class 4: F1-score of 0.03, indicating poor performance.
    - Class 6: F1-score of 0.06, indicating poor performance.
    - Class 7: F1-score of 0.08, indicating poor performance.
    - Class 9: F1-score of 0.04, indicating poor performance.
    - Class 11: F1-score of 0.00, indicating poor performance.
2. TF-IDF Vectorizing with preprocessing:
    - Class 4: F1-score of 0.02, indicating poor performance.
    - Class 6: F1-score of 0.06, indicating poor performance.
    - Class 7: F1-score of 0.07, indicating poor performance.
    - Class 9: F1-score of 0.04, indicating poor performance.
    - Class 11: F1-score of 0.00, indicating poor performance.
3. Count Vectorizer without preprocessing:
    - Class 4: F1-score of 0.06, indicating poor performance.
    - Class 7: F1-score of 0.08, indicating poor performance.
    - Class 9: F1-score of 0.09, indicating poor performance.
4. Count Vectorizer with preprocessing:
    - Class 4: F1-score of 0.06, indicating poor performance.
    - Class 7: F1-score of 0.06, indicating poor performance.
    - Class 9: F1-score of 0.09, indicating poor performance.
5. Glove without preprocessing:
6. Glove with preprocessing:
    - Class 4: F1-score of 0.06, indicating poor performance.

Artificial Neural Network (ANN - Sequential):

1. 13 classes without preprocessing: Class 3, Class 4, Class 6, Class 7, Class 9, and Class 11 have low F1-scores of 0.00, 0.00, 0.00, 0.00, 0.05, and 0.00, respectively.
2. 13 classes with preprocessing: Class 3, Class 4, Class 6, Class 7, and Class 9 have low F1-scores of 0.00, 0.00, 0.00, 0.00, and 0.06, respectively.

Transformers (BERT, PubMedBERT, GPT2):

- For these transformer-based models, several classes have low F1-scores, but there isn't a specific pattern as in the previous models. The classes with low F1-scores vary depending on the specific model, preprocessing, and vectorization technique. Classes with low F1-scores should be analyzed individually to understand the reasons for poor performance.
- In the transformer models, certain problematic classes, such as 4, 9, and 11, continue to pose challenges for prediction, while classes 2 and 3 also exhibit difficulties in certain cases.
- Notably, in the "pubmedbert" model without preprocessing, it is worth mentioning that nearly all classes are predicted effectively, with the exception of class 4, which encounters issues. Another point to consider is that, after preprocessing, the model generally performs better, but classes 2 and 9 still experience classification deficiencies.
- For a more in-depth analysis and addressing these issues to enhance performance, it is advisable to individually investigate samples from the dataset, segregating them as needed.

summary:

there is an imbalance in class distribution. Some classes have significantly fewer samples than others. This can lead to biased model predictions and affect model performance. Recall is an important metric, especially when dealing with imbalanced datasets. In many cases, the recall values are low, indicating that the models are not effectively capturing all instances of certain classes. This is likely due to class imbalance. Precision values are also relatively low for some classes, indicating that the models are making false positive predictions. This could be due to noisy labels or challenges in distinguishing between certain classes.

Preprocessing seems to have some impact on model performance, but it doesn't drastically improve the results. Class imbalance and noisy labels may still persist even after preprocessing.

Different vectorization techniques (TF-IDF, Count Vectorizer, GloVe, BERT, GPT2) have varying impacts on model performance. Some techniques may perform better for certain tasks, but overall, the models are facing challenges with the given dataset.

Future work:

In the realm of enhancing model performance, particularly when confronted with class imbalance and noisy labels, there exist several promising avenues for future exploration. It is essential to acknowledge that the task of refining both transformer models and artificial neural networks (ANNs) warrants greater attention, especially in terms of extended training epochs. However, the scope of our current study was somewhat constrained by limited time and the pressing need to evaluate a multitude of models, with particular consideration for the demanding training requirements of transformer architectures.

Looking ahead, here are some prospective directions for further research and improvement:

1. Ensemble Methods: Ensembles like Random Forest or Gradient Boosting offer a potent means of ameliorating class imbalance while simultaneously elevating overall model performance. They exhibit a commendable ability to manage noisy labels more adeptly than individual models.

2. Data Augmentation: One viable strategy involves data augmentation, particularly for classes with meager sample sizes. By synthetically increasing the volume of training data, a more balanced class distribution can be achieved.

3. Resampling Techniques: Overcoming class imbalance can also be pursued through resampling techniques. Oversampling the minority classes or undersampling the majority classes can help rebalance the dataset.

4. Noise Reduction: A meticulous data review, coupled with judicious cleaning, can effectively minimize noisy labels and rectify inconsistencies in the dataset.

5. Hyperparameter Tuning: The optimization of model hyperparameters represents a crucial dimension. Fine-tuning these parameters can be instrumental in achieving peak performance, though this endeavor often demands substantial computational resources.

6. Feature Engineering: Delving into feature engineering techniques can yield more informative features, consequently augmenting the discriminative power of the models.

7. Data Expansion: Where feasible, expanding the dataset, especially for underrepresented classes, can be pivotal in addressing the challenges associated with class imbalance and noisy data.

8. Transfer Learning: Depending on the nature of the task at hand, exploring transfer learning using pre-trained models can be advantageous. This approach allows you to harness the wealth of knowledge stored in larger datasets.

9. Misclassification Analysis: A thorough examination of misclassified samples can illuminate the specific hurdles and patterns that our models grapple with. This knowledge can guide subsequent improvements.

It's important to bear in mind that elevating model performance in the face of imbalanced and noisy datasets is a multifaceted undertaking. Success hinges on the synergy of data preprocessing, feature engineering, and model fine-tuning. Though the road may be challenging, the potential rewards in terms of model robustness and accuracy are well worth the effort. As we embark on this journey of refinement, the optimization of transformer models and ANNs remains an intriguing avenue for further exploration and advancement.

Conclusion:

In summary, our observations indicate that preprocessing has had an overall positive impact on the obtained results, significantly improving model performance. The best results were achieved when employing text data preprocessing techniques. Furthermore, reducing the number of labels from 40 to 13, albeit introducing a slight imbalance in the labels, led to substantial performance improvements. Hence, it is evident that label imbalance is one of the factors contributing to model performance degradation, and it is recommended to address this issue for improved results. A suitable approach for tackling this issue is data augmentation.

To further enhance model performance, we recommend addressing label imbalance, potentially through data augmentation techniques. Additionally, acquiring a larger training dataset is advisable. Notably, the best results were obtained using the "pubmedbert" model, a fine-tuned transformer model designed for medical datasets. Transformer models require substantial amounts of data for training, and the prior training of "pubmedbert" on medical data likely contributed to its superior performance. Therefore, it is expected that substantial improvements can be achieved by increasing the dataset size using various available techniques.

Considering that the average text length in the dataset was approximately 500 tokens, segmenting and separating these sections to generate new records, as well as employing techniques such as summarization and sequence-to-sequence models, can be utilized to both mitigate label imbalance issues and enlarge the dataset, thereby facilitating better model training.

Furthermore, the limited training epochs, particularly five epochs for transformer models, were dictated by time constraints. It is anticipated that in a more in-depth study, increasing the number of epochs for these models will yield superior results.

In conclusion, our study underscores the potential for substantial improvements in model performance through enhanced preprocessing, addressing label imbalance, and augmenting the dataset. Models like "pubmedbert" exhibit promise in medical domain tasks, emphasizing the significance of pre-training on relevant data. Future research endeavors should focus on expanding the dataset using various techniques and conducting more extensive training, particularly for transformer models, to further refine and elevate model performance.