

# NLP M1 Report

Omar Aboelazm  
Mohammed Assem

April 23, 2024

## Abstract

In the realm of Natural Language Processing (NLP), the ability to classify text accurately and effectively has become a fundamental task with widespread applications. From sentiment analysis to information retrieval, the capacity to decipher textual data plays a pivotal role in numerous domains. Our project endeavors to delve into this realm by addressing the task of question classification and answering system, utilizing real-world datasets sourced from two prominent StackExchange forums: Android Enthusiasts and Ask Different (Apple).

## 1 Introduction

Our selection of the "Android vs. iOS" dataset stemmed from a strategic consideration of relevance, accessibility, and potential for exploration. The dataset offers a rich repository of questions posted by users on both Android and iOS platforms, enabling us to conduct comprehensive analyses and develop sophisticated models. By leveraging this dataset, we aim to tackle a significant challenge in NLP—differentiating between questions pertaining to Android and iOS platforms and providing accurate responses.

### 1.1 Relevance and Significance:

The ubiquity of mobile technology, particularly the Android and iOS ecosystems, underscores the importance of understanding user queries and preferences within these domains. As mobile devices continue to shape various aspects of modern life, the ability to categorize and address user inquiries accurately holds immense practical value. Our project seeks to contribute to this endeavor by employing advanced NLP techniques to classify questions and build an effective question-answering system.

### 1.2 Challenges and Opportunities:

The task of question classification and answering system presents several challenges, including handling unstructured text data, mitigating class imbalances, and ensuring model robustness across diverse query types. However, these challenges also represent opportunities for innovation and learning. By navigating through these complexities, we aim to gain insights into the intricacies of text classification and contribute to the development of practical solutions that enhance user experiences in the mobile technology domain.

### 1.3 Project Objectives:

In the subsequent sections of this report, we will delve into a comprehensive literature review, exploring recent research relevant to our problem domain. Additionally, we will conduct a thorough analysis of the dataset at hand, uncovering insights and potential challenges that will inform our approach in the subsequent milestones.

## 2 Literature Review

Recent advancements in question answering (QA) systems have showcased the potential of natural language processing (NLP) techniques in automating information retrieval and knowledge dissemination. (Lavanya, 2021). Traditional approaches, primarily relying on keyword matching and syntactic search algorithms, have paved the way for more sophisticated methodologies that prioritize semantic understanding and context-based responses (Beta et. al, 2023). However, challenges persist, particularly in low-resource languages and specialized domains such as mobile technology platforms like Android and iOS.

### 2.1 Addressing Challenges in Low-Resource Languages

Das and Saha (2022) address the challenge of building QA systems in low-resource languages like Bengali by employing supervised learning algorithms and leveraging machine-readable dictionaries such as WordNet. Their system achieves high accuracy in question classification and answer retrieval, demonstrating the feasibility of building comprehensive QA systems in languages with limited linguistic resources. The adoption of supervised learning methods and the use of a text corpus as the system's repository underscore the importance of utilizing available resources effectively in overcoming language-specific challenges.

### 2.2 Harnessing Deep Learning and NLP Techniques

Tzu-Hsuan Lin et al. (2022) leverage deep learning and NLP techniques, specifically the Bidirectional Encoder Representations from Transformers (BERT) model, to develop an intelligent question and answer system tailored to the construction industry's needs. By integrating BERT with a mobile chatbot interface, their system enables conversational machine understanding and facilitates user-friendly information searches in the context of building information modeling and artificial intelligence of things (BIM-AIOT). The utilization of machine learning models and NLP techniques enables accurate prediction and efficient information retrieval, empowering professionals in the Architecture, Engineering, and Construction (AEC) domain to make informed decisions swiftly.

### 2.3 Pipelines for QA and QC Models

In Natural Language Processing (NLP), several common pipelines and model architectures have emerged as standard approaches for various tasks. One prevalent pipeline involves preprocessing the text data by tokenizing, normalizing, and vectorizing the text before feeding it into a neural network-based model. For classification tasks, Convolutional Neural Networks (CNNs) and recurrent architectures like Long Short-Term Memory (LSTM) networks are widely used due to their ability to capture sequential information effectively.

Attention mechanisms have become a fundamental component in many state-of-the-art models across various fields of natural language processing (NLP). Originally introduced in the context of neural machine translation (Bahdanau et al., 2014), attention mechanisms allow models to focus on relevant parts of the input sequence when making predictions. The key idea is to dynamically weigh different parts of the input sequence, giving higher importance to the more relevant tokens. They have since been adopted in a wide range of NLP tasks, including text classification, language modeling, and question answering. They have also been extended and refined in various ways, such as self-attention mechanisms in transformer models (Vaswani et al., 2017), which enable capturing complex dependencies between tokens in the input sequence.

Another common practice in NLP research is the use of transfer learning and pre-trained language models. Researchers often leverage pre-trained models like BERT (Devlin et al., 2018), GPT (Radford et al., 2018), or RoBERTa (Liu et al., 2019) as starting points for their specific NLP tasks. These pre-trained models are fine-tuned on domain-specific data or downstream tasks to adapt them to the specific task at hand. This approach has proven effective in achieving state-of-the-art results across a wide range of NLP tasks, including text classification, named entity recognition, question answering, and machine translation.

Muhammad Zulqarnain et al. (2021) compared different neural networks architectures for QC, and among the tested layers which were CNN, LSTM and GRU, the combination of CNN layer followed

by an GRU layer yielded the best results.

These pipelines and architectures, along with their variations and combinations, form the backbone of modern NLP research and applications.

## 2.4 Implications for Android vs. iOS Question Classification

While the aforementioned studies focus on diverse domains such as language-specific QA systems and BIM-AIOT integration, their methodologies and insights offer valuable lessons for the task of classifying questions related to Android and iOS support platforms. By adopting supervised learning algorithms, deep learning models, and NLP techniques, researchers can enhance the accuracy and robustness of question classification systems, enabling more effective differentiation between user queries in the mobile technology domain. Furthermore, the utilization of large text corpora and domain-specific knowledge repositories can facilitate the development of comprehensive QA systems tailored to the intricacies of Android and iOS platforms.

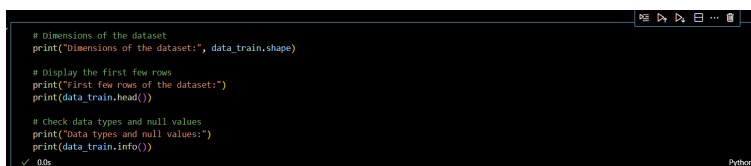
## 2.5 Conclusion

In conclusion, recent advancements in QA systems, exemplified by the works of Das and Saha (2022) and Tzu-Hsuan Lin et al. (2022), demonstrate the transformative potential of deep learning, NLP, and supervised learning techniques in automating information retrieval and knowledge dissemination across diverse domains. By leveraging these methodologies and insights, researchers can tackle the challenges of question classification in specialized domains such as Android and iOS platforms, paving the way for more accurate and efficient question answering systems tailored to the needs of modern mobile technology users.

# 3 Data analysis and insights

For the following part, we will be discussing and analyzing all aspects of the dataset, while also extracting as many insights as possible.

## 3.1 Value analysis



```
# Dimensions of the dataset
print("Dimensions of the dataset:", data_train.shape)

# Display the first few rows
print("First few rows of the dataset:")
print(data_train.head())

# Check data types and null values
print("Data types and null values:")
print(data_train.info())
```

Figure 1: Code Snippet

Starting with the basics, we used simple pandas methods to view the dataset dimensions, the first few rows, the data types used and null occurrences. As shown, the dataset has 51370 entries and 7 columns. Luckily, this dataset is clean from null values.

The following is just a clearer sample of the dataset.

As seen, the seven columns are:

- Id: question id, each question has a unique value.
- Title: title of the question asked.
- Body: questions body related to the title.
- Score: score assigned to the the questions.
- ViewCount: number of views the question got.
- Label: classification label, whether the question is related to android or ios



```

import matplotlib.pyplot as plt

# Distribution of Labels
label_distribution = data_train['Label'].value_counts()
print("Distribution of Labels:")
print(label_distribution)

# Pie chart of label distribution
plt.figure(figsize=(6, 6))
plt.pie(label_distribution, labels=label_distribution.index, autopct='%1.1f%%')
plt.title("Distribution of Labels")
plt.show()

```

✓ 0.0s Python

Figure 5: Code Snippet

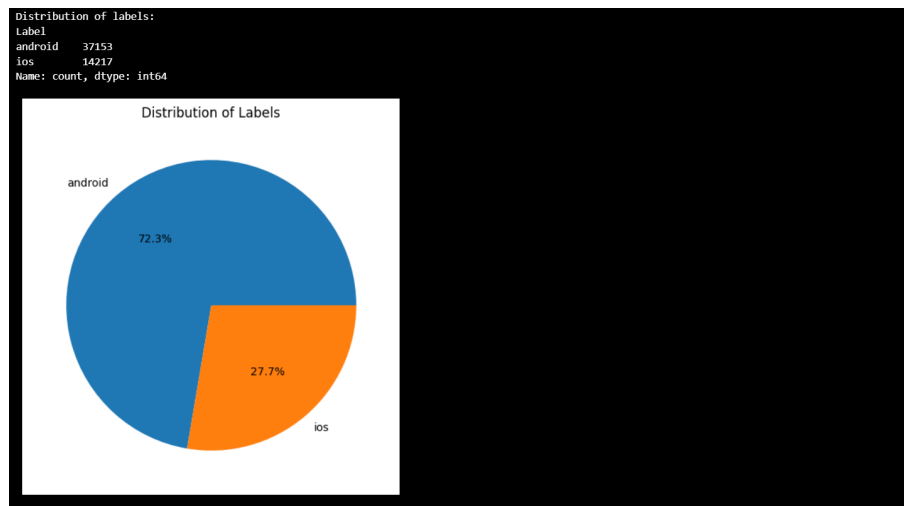


Figure 6: Label Distribution

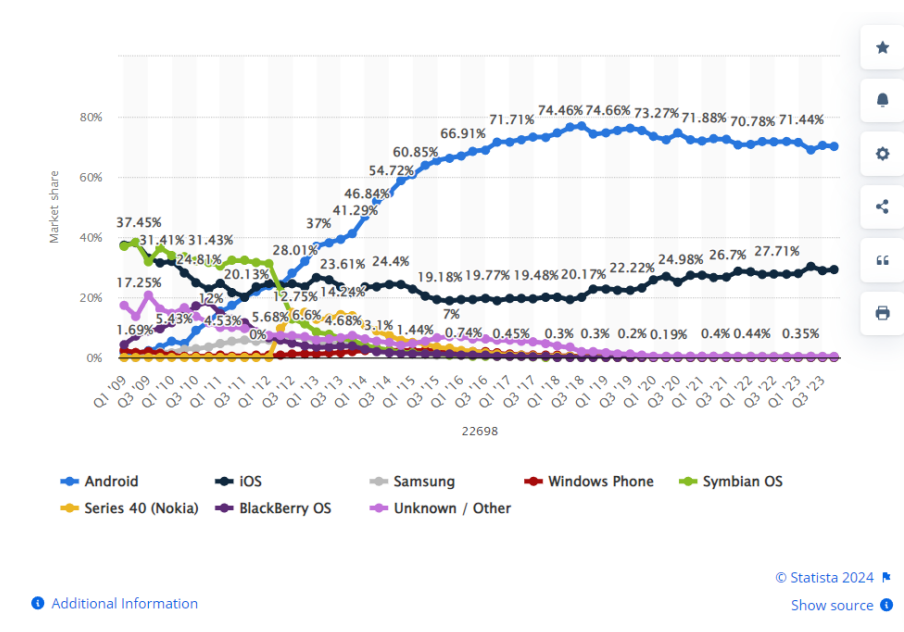


Figure 7: Statista Data

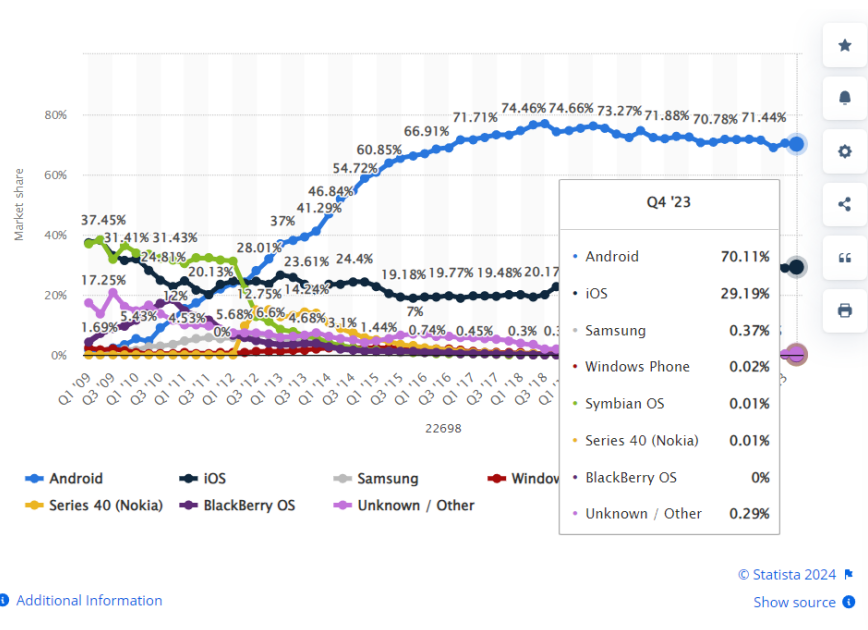


Figure 8: Statista Data

But, this bias is to be considered during the training of the model.

- Score distribution

```
import seaborn as sns

# Define function to remove outliers based on IQR
def remove_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return data[(data[column] >= lower_bound) & (data[column] <= upper_bound)]

# Visualize distribution of 'Score' before removing outliers
plt.figure(figsize=(8, 6))
sns.boxplot(x=data_train['Score'])
plt.title('Distribution of Score (Before Removing Outliers)')
plt.xlabel('Score')
plt.show()

# Remove outliers from 'Score' column
data_no_outliers = remove_outliers_iqr(data_train, 'Score')

# Visualize distribution of 'Score' after removing outliers
plt.figure(figsize=(8, 6))
sns.boxplot(x=data_no_outliers['Score'])
plt.title('Distribution of Score (After Removing Outliers)')
plt.xlabel('Score')
plt.show()
```

Figure 9: Box-plot Distribution

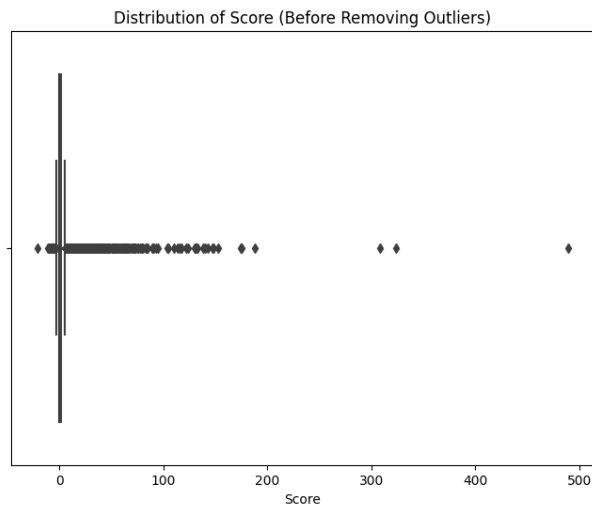


Figure 10: With Outliers

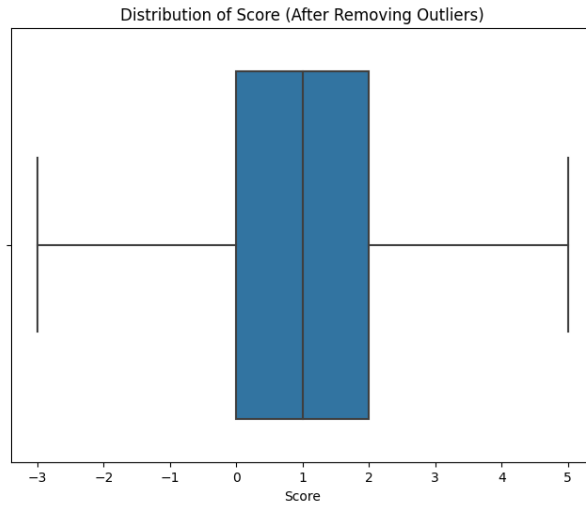


Figure 11: Without Outliers

The first box-plot shows the distribution of score before removing outliers, and the right box-plot shows the distribution of values after removing outliers. Outliers were removed using the Interquartile Range (IQR) method. It is a measure of statistical dispersion, or spread, which is used to identify the extent of spread in the middle 50% of a dataset. The IQR is calculated as the difference between the third quartile (Q3) and the first quartile (Q1) of the data.

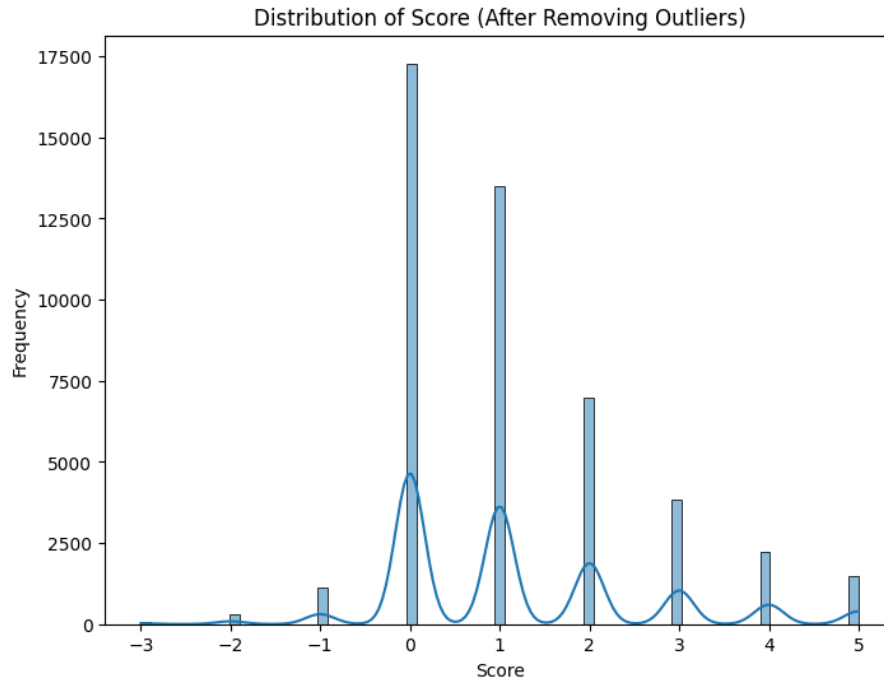


Figure 12: Histogram Without Outliers

The above histogram visualizes the distribution of scores after removing outliers.

- **Word-cloud**



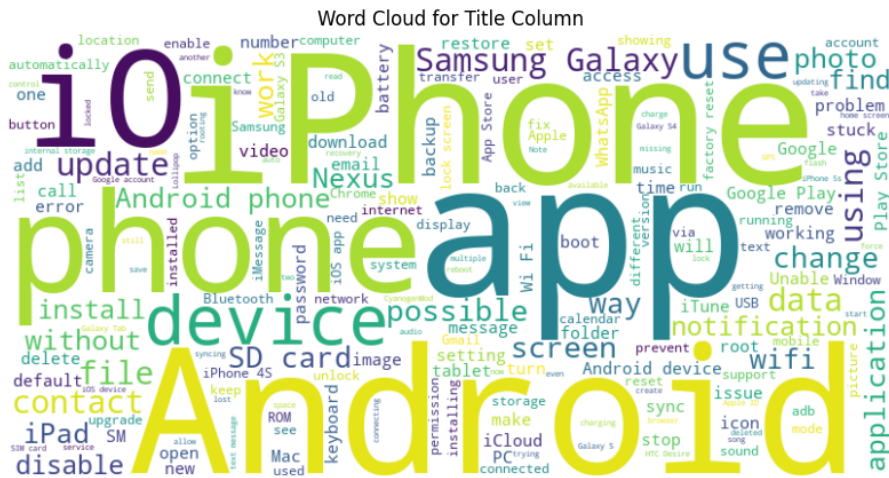


Figure 13: General WordCloud



Figure 14: Specific WordCloud

Word-cloud was used in the first image to visualize the most used words in the “Title” column, and then re-used to visualize the words related to each OS class separately.

- Sentiment polarity analysis

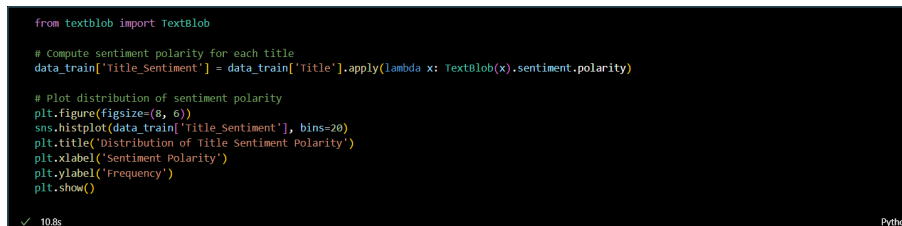


Figure 15: Sentiment Analysis

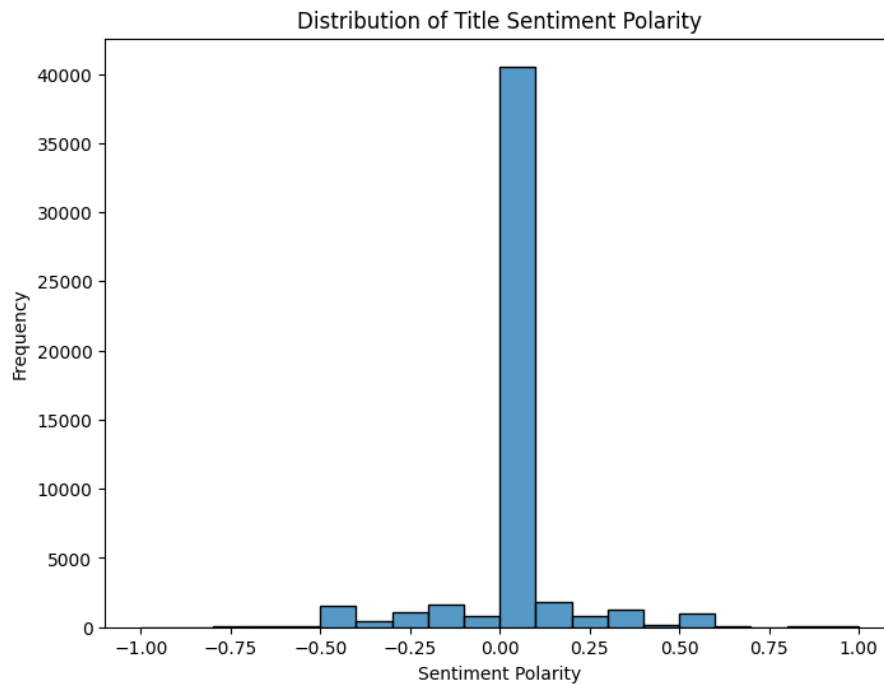


Figure 16: Sentiment Distribution

The above plot shows that most questions have a sentiment polarity of 0, which is neutral. That is logical as these questions are mostly technical.

- **Topic classification**

```
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer

# Vectorize text data
vectorizer = CountVectorizer(max_features=1000, stop_words='english')
X = vectorizer.fit_transform(data_train["title"])

# Apply LDA
lda = LatentDirichletAllocation(n_components=5, random_state=42)
lda.fit(X)

# Display top words for each topic
feature_names = vectorizer.get_feature_names_out()
for topic_idx, topic in enumerate(lda.components_):
    print(f"Topic {topic_idx+1}:")
    print(" ".join([feature_names[i] for i in topic.argsort()[::-10:-1]]))
    print()
```

Figure 17: Topic Classification

```
Topic 1:
galaxy samsung google wifi does android work doesn network internet

Topic 2:
app android device apple ios music does keyboard change htc

Topic 3:
android data phone contacts device account using adb sync file

Topic 4:
iphone screen phone lock messages notification text android sms usb

Topic 5:
ios apps app card android play sd google iphone store
```

Figure 18: Topics

```
Topic 1: Connectivity and Network Issues (Android):
This topic includes terms related to connectivity and network problems commonly experienced by Android users, such as issues with Wi-Fi, internet, and network connectivity.

Topic 2: App and Device Management (Android and iOS):
This topic covers terms related to managing apps and devices, including topics like app installation, device compatibility, and keyboard settings. It appears to encompass aspects relevant to both Android and iOS platforms.

Topic 3: Data Management and Synchronization (Android):
This topic focuses on data management and synchronization tasks associated with Android devices. It includes terms related to managing phone data, syncing contacts, and using tools like ADB (Android Debug Bridge).

Topic 4: Device Operations and Notifications (Android and iOS):
This topic discusses various device operations and notifications, including tasks like locking the screen, receiving text messages, and handling notifications. These operations are relevant to both Android and iOS platforms.

Topic 5: App Store and Application Management (iOS and Android):
This topic revolves around app store-related terms and application management tasks, including topics like installing apps, using app stores (e.g., Google Play, App Store), and managing storage space with SD cards. It appears to encompass aspects relevant to both iOS and Android platforms.
```

Figure 19: Topics

We utilized Latent Dirichlet Allocation (LDA) to uncover latent topics within text data. We first vectorized the text using CountVectorizer, then trained the LDA model to identify topics based on word distributions. Finally, we displayed the top words associated with each topic, providing insight into the main themes captured by the model.

### 3.3 Data pre-processing

#### Lowercasing

```
Data Preprocessing
+ Code + Markdown

Lowercasing

data_train['Title'] = data_train['Title'].str.lower()
data_train['Body'] = data_train['Body'].str.lower()

✓ 0.1s Python
```

Figure 20: Lowercasing

The lowercasing is done to normalize the words for future tokenization and model training.

### 3.4 Tokenization

```

from transformers import BertTokenizer
from bs4 import BeautifulSoup
import pandas as pd
import re

# Initialize the tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Custom function to clean text from HTML links and tags, then tokenize text while preserving version numbers and contractions
def preprocess_and_tokenize(text):
    # Remove HTML tags
    cleaned_text = BeautifulSoup(text, "html.parser").get_text(separator=" ")

    # Tokenize text while preserving version numbers and contractions
    tokens = []
    current_word = ""
    for char in cleaned_text:
        if char.isalnum() or char in ["'", "-", "."]:
            current_word += char
        else:
            if current_word:
                tokens.append(current_word)
                current_word = ""
            if char.strip(): # If char is not empty, add it to tokens
                tokens.append(char)
    if current_word:
        tokens.append(current_word)
    return tokens

# Apply preprocessing and tokenization to the 'Title' and 'Body' columns
data_train['title_tokens'] = data_train['title'].apply(preprocess_and_tokenize)
data_train['body_tokens'] = data_train['body'].apply(preprocess_and_tokenize)

# Print the DataFrame to verify preprocessing
print(data_train[['title', 'title_tokens']].head())
print(data_train[['body', 'body_tokens']].head())

```

Figure 21: Tokenization Process

```

                                title
0 drop\stop mobile data connection (non-wifi) by... \
1 how to automatically crop text messages when s...
2 can't find text message that was to a group
3 can't store contacts on my android phone
4 dropbox on samsung galaxy - where is the setti...

                                title_tokens
0 [drop, \, stop, mobile, data, connection, (, n...
1 [how, to, automatically, crop, text, messages,...
2 [can't, find, text, message, that, was, to, a,...
3 [can't, store, contacts, on, my, android, phone]
4 [dropbox, on, samsung, galaxy, -, where, is, t...

                                Body
0 <p>can i set android 4.4.2 to drop mobile data... \
1 <p>is there a way to prevent the messages app ...
2 <p>when john doe texts to a group that include...
3 <p>i was going through all of my installed app...
4 <p>on a sony xperia, the settings button in dr...

                                Body tokens
0 [can, i, set, android, 4.4.2, to, drop, mobile...
1 [is, there, a, way, to, prevent, the, messages...
2 [when, john, doe, texts, to, a, group, that, i...
3 [i, was, going, through, all, of, my, installe...
4 [on, a, sony, xperia, ,, the, settings, button...

```

Figure 22: Tokenization Output

Tokenization is done to the dataset after lowercasing, it produces a list of all the words used in the dataset.

### 3.5 Removing punctuation

```

Removing Punctuation

import string

# Remove punctuation from title and body columns
data_train['title_tokens'] = data_train['title_tokens'].apply(lambda x: [word for word in x if word not in string.punctuation])
data_train['body_tokens'] = data_train['body_tokens'].apply(lambda x: [word for word in x if word not in string.punctuation])

# Print the DataFrame to verify preprocessing
print(data_train[['title', 'title_tokens']].head())
print(data_train[['body', 'body_tokens']].head())

```

Figure 23: Removing punctuation

Cleaning the tokens by removing punctuations.

### 3.6 Removing Stopwords



```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

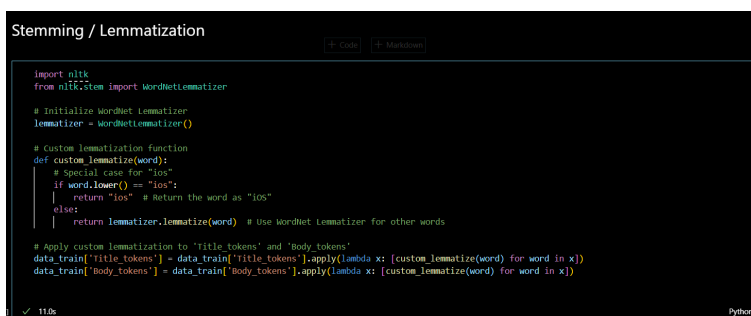
# Remove stopwords from title and body columns
data_train['title_tokens'] = data_train['title_tokens'].apply(lambda x: [word for word in x if word not in stop_words])
data_train['body_tokens'] = data_train['body_tokens'].apply(lambda x: [word for word in x if word not in stop_words])

# Print the DataFrame to verify preprocessing
print(data_train[['title', 'title_tokens']].head())
print(data_train[['body', 'body_tokens']].head())
```

Figure 24: Removing Stopwords

Cleaning the tokens by removing stop words that will not add much to the context.

### 3.7 Stemming / Lemmatization



```
import nltk
from nltk.stem import WordNetLemmatizer

# Initialize wordnet lemmatizer
lemmatizer = WordNetLemmatizer()

# Custom lemmatization function
def custom_lemmatize(word):
    # Special case for "ios"
    if word.lower() == "ios":
        return "ios" # Return the word as "ios"
    else:
        return lemmatizer.lemmatize(word) # Use WordNet Lemmatizer for other words

# Apply custom lemmatization to 'title_tokens' and 'body_tokens'
data_train['title_tokens'] = data_train['title_tokens'].apply(lambda x: [custom_lemmatize(word) for word in x])
data_train['body_tokens'] = data_train['body_tokens'].apply(lambda x: [custom_lemmatize(word) for word in x])

data_train.head()
```

Figure 25: Stemming / Lemmatization

Returning the tokens to their stem word.

### 3.8 Joining tokens



```
data_train['title_clean'] = data_train['title_tokens'].apply(lambda x: ' '.join(x))
data_train['body_clean'] = data_train['body_tokens'].apply(lambda x: ' '.join(x))

data_train.head()
```

Figure 26: Joining tokens

Reconstructing the sentences by joining the final tokens.

### 3.9 Token Analysis

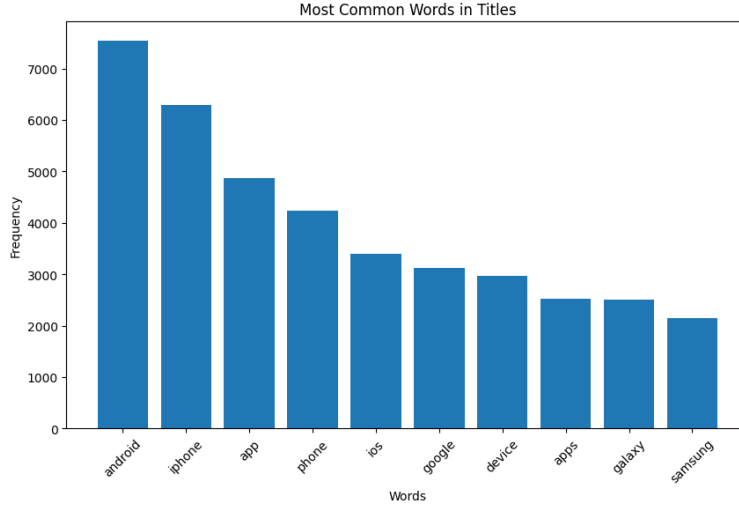


Figure 27: Title tokens

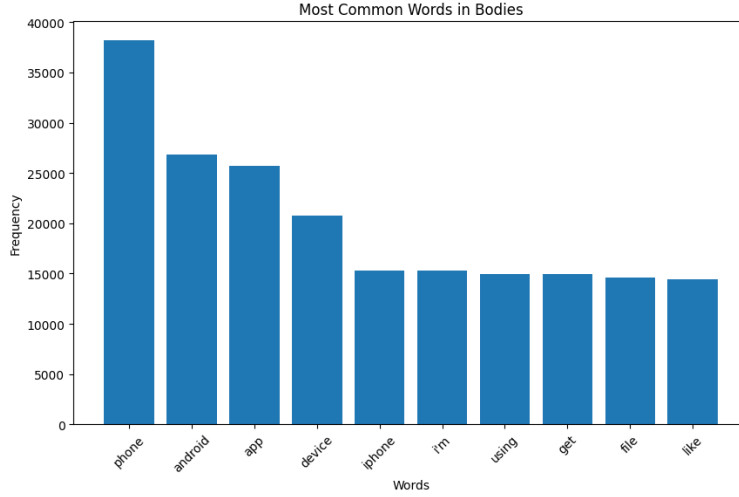


Figure 28: Body tokens

After applying all the cleaning methods for tokens, the above histogram visualizes the most frequent tokens in both the Body and Title columns. Which shows a successful tokenization as these words are mostly associated with technical questions related to android and ios.

### 3.10 Critical Insights

1. Imbalance in Android vs. iOS Content: The dataset heavily leans towards Android-related questions, reflecting the dominance of Android in the market compared to iOS. This skew raises concerns about potential differences in user interests or available information, highlighting the need for careful model training to maintain balance.
2. Effect of Extreme Values on Scores: Removing outliers significantly changes the distribution of scores, indicating that extreme values play a significant role in shaping overall trends. Understanding these outliers' nature and impact is crucial for accurate modeling and interpretation.
3. Digging Deeper into Word Frequency: While word clouds provide initial insights into common themes, a deeper analysis of word frequency could reveal underlying patterns and topics. Uncovering and understanding these subtleties can enhance analysis and model performance.

4. Exploring Neutral Sentiment: The predominance of neutral sentiment reflects the dataset's technical nature. However, examining sentiment fluctuations and their relationship with user engagement could uncover valuable insights for optimizing content and enhancing user satisfaction.
5. Ensuring Data Integrity through Pre-processing: Thorough pre-processing techniques, such as advanced tokenization to maintain sequence integrity and handling contractions, are essential for preserving data accuracy and minimizing information loss. Multiple problems were encountered as contraction tokenization and version tokenization (e.g. 4.4.2). Which needed special and advanced tokenization.

In summary, examining label biases, understanding outlier effects, conducting nuanced word frequency analysis, exploring sentiment trends, and employing meticulous pre-processing practices are crucial for extracting valuable insights and ensuring reliable model performance in real-world datasets.

## 4 Classification Model

Initially, the task at hand involved classifying text data into binary categories (1 = IOS & 0 = ANDROID) using a shallow neural network that was not pre-trained. The dataset consisted of textual information of questions related to both categories split into titles and bodies, each associated with a numerical label that identified the class. The goal was to predict these labels based on the textual content.

### 4.1 Data Preprocessing

Starting with data pre-processing, the textual data was tokenized using the Tokenizer module from Keras, to transform the string tokens into sequence of integers. Both title and body data were combined for each entry, creating a single sequence which served as a wider feature for learning the class. This combined sequence was then split into training and testing sets using a standard 80-20 split. Additionally, padding was applied to ensure uniform sequence length for model input, with a maximum sequence length of 100 tokens.

```
Raw data tokens: ['twrp', 'recovery', 'android', 'doubt', 'twrp', 'recovery', 'tool', 'developed', 'smartphone', 'specifically', 'even', 'almost',
Numerically embedded data tokens: [270, 89, 2, 2180, 270, 89, 383, 2458, 609, 903, 52, 586, 123, 2, 1]
Padded data tokens: [ 270   89    2 2180  270   89  383 2458  609  903   52  586  123    2
1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0]
```

Figure 29: Tokenization and Embedding

Also the dataset at hand, has an imbalance between the classes as previously discussed. To address this imbalance, class weights were computed and incorporated into the model training process using the `compute_class_weight` function from scikit-learn. These class weights were utilized to provide higher importance to minority class samples during training, thus mitigating the imbalance issue.

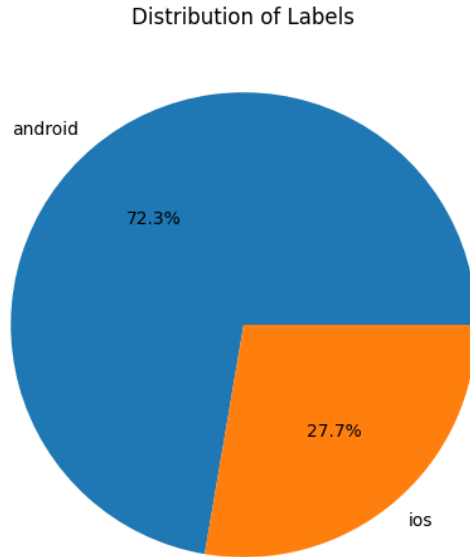


Figure 30: Label Distribution

## 4.2 Model Architecture

Subsequently, we chose an RNN model. First, a simple RNN model from Keras was used to limit the computational impact on our devices as we are dealing with a medium sized dataset. But, it didn't perform well and only reached an accuracy of 52%

Then, a Long Short-Term Memory (LSTM) network, which is more complex, was used. The model architecture included an embedding layer for word representation, followed by an LSTM layer for sequential processing, and a dense layer with sigmoid activation for binary classification.

Lastly, a more complicated and dynamic architecture was created with one embedding layer, three bi-directional LSTM layers and a dense layer. The input is embedding generated using Word2Vec embeddings to try and achieve a better accuracy.

The LSTM and BI-LSTM had the same accuracy of 98% but the BI-LSTM had slightly better precision, recall and F1-Score values. Which indicates it has lower false positives and false negatives.

So, after multiple model architectures we stuck with the BI-LSTM NN.



```

# Word2Vec model for embeddings
word2vec_model = Word2Vec(sentences=X_combined, vector_size=embedding_dim, window=5, min_count=1, workers=4)

# Model architecture
class_model = Sequential()

# Embedding layer
class_model.add(Embedding(input_dim=len(class_tokenizer.word_index) + 1,
                          output_dim=embedding_dim,
                          weights=[np.vstack([np.zeros(embedding_dim), word2vec_model.wv.vectors])],
                          input_length=maxlen, trainable=False))

# Dynamic sized bidirectional LSTM layers
for _ in range(num_lstm_layers):
    class_model.add(Bidirectional(LSTM(units=lstm_units, dropout=0.2, recurrent_dropout=0.2, return_sequences=True)))

# Final bidirectional LSTM layer
class_model.add(Bidirectional(LSTM(units=lstm_units, dropout=0.2, recurrent_dropout=0.2)))

# Dense layer with sigmoid AF
class_model.add(Dense(1, activation='sigmoid'))

class_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

```

Figure 31: Architecture 2

### 4.3 Model Training & Evaluation

The model was compiled with the Adam optimizer and binary cross-entropy loss function. Adam optimizer was used due its dynamic nature and robustness.

The BI-LSTM model was trained using a batch size of 32 and for a specified number of epochs (in this case, 10 epochs). During training, a validation split of 10% was utilized for monitoring model performance and preventing overfitting. Following training, the model was evaluated on the test set to assess its accuracy in predicting binary labels.

```

# Hyperparameters
maxlen = 100
embedding_dim = 300 # Dimensionality of Word2Vec embeddings
lstm_units = 128
num_lstm_layers = 2 # Number of BI-LSTM layers
batch_size = 32
epochs = 10

```

Figure 32: Hyperparameters

Model performance was evaluated using various metrics, including accuracy and confusion matrix analysis. The confusion matrix provided insights into the model's classification performance by visualizing true positive, true negative, false positive, and false negative predictions.

Class	Precision	Recall	F1-Score	Support
0	0.98	0.99	0.99	7318
1	0.97	0.96	0.97	2956
<b>Accuracy</b>			0.98	10274
<b>Macro Avg</b>	0.98	0.98	0.98	10274
<b>Weighted Avg</b>	0.98	0.98	0.98	10274

Table 1: Classification report

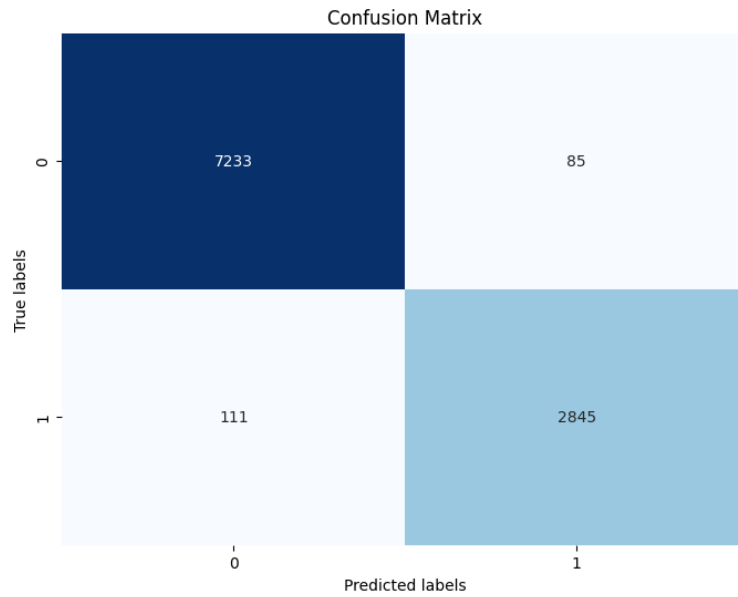


Figure 33: Confusion Matrix

Due to handling the imbalances and the appropriate size of the dataset, acceptable accuracies were achieved by the BI-LSTM Network. The model was then tested on custom user inputs to validate its usage.

```

user_input = input("Enter: ")
user_input_seq = class_tokenizer.texts_to_sequences([user_input])
user_input_pad = pad_sequences(user_input_seq, maxlen=100, padding='post', truncating='post')
print("User input: ", user_input)
predicted_label = class_model.predict(user_input_pad)
predicted_label_binary = 1 if predicted_label > 0.5 else 0
predicted_label_st = "Android" if predicted_label_binary == 0 else "IOS"

print("Predicted Label: ", predicted_label_st)
✓ 63s

User input: I love samsung but have an iphone
1/1 [=====] - 0s 37ms/step
Predicted Label: IOS

```

Figure 34: User Input

```

user_input = input((variable) class_tokenizer: Tokenizer)
user_input_seq = class_tokenizer.texts_to_sequences([user_input])
user_input_pad = pad_sequences(user_input_seq, maxlen=100, padding='post', truncating='post')
print("User input: ", user_input)
predicted_label = class_model.predict(user_input_pad)
predicted_label_binary = 1 if predicted_label > 0.5 else 0
predicted_label_st = "Android" if predicted_label_binary == 0 else "IOS"

print("Predicted Label: ", predicted_label_st)
✓ 8.6s

User input: I love apple but have a samsung
1/1 [=====] - 0s 33ms/step
Predicted Label: Android

```

Figure 35: User Input

## 4.4 Conclusion

Summing up the work, we successfully created and trained a classification model able to classify a given question to either relating to IOS or Android. The features where the title and body of questions asked on Stack Exchange and the target feature was the label given to these questions by users on the same platform.

After testing multiple architectures, it was clear that the BI-LSTM was fit for the task at hand. The only drawbacks were that the model was not pre-trained and didnt have any data labeled as "Irrelevant". This will cause the model to ultimately predict a label of "IOS" or "Android" for any input even if it was completely irrelevant to both subjects. Yet, the model had a satisfactory performance with both labels.

Future work will include using a pre-trained model and trying more complex architectures and further tune the hyper-parameters to achieve better performance.

## 5 Regression Model

In this project, our objective was to develop a regression model capable of accurately predicting scores based on the textual content from the questions of IOS Vs Android dataset. The dataset used for this task contains various features such as the title, body, view count, and score of each question. Our goal was to leverage natural language processing (NLP) techniques and machine learning algorithms to accurately predict the score of a question based on its textual content. This report provides a detailed overview of the data preprocessing steps, model architecture, training process, evaluation metrics, and rationale behind key design choices.

### 5.1 Data Preprocessing

The initial phase of the project involved thorough data preprocessing to ensure the quality and reliability of the dataset. We performed an exploratory analysis to identify any inconsistencies or anomalies within the data after and including some of that was held in milestone 1. One crucial concern we had was the presence of entries with a view count of 0 but a positive score, which contradicted typical behavior. To maintain data integrity, we deemed these entries as invalid and removed them from the dataset to prevent potential noise during model training. Fortunately, when we checked the dataset we found no such entries.

```
# Check for entries with viewCount = 0 and score > 0
invalid_entries = reg_data[(reg_data['ViewCount'] == 0) & (reg_data['Score'] > 0)]

invalid_entries
✓ 1.2s
```

Id	Title	Body	Score	ViewCount	Label	LabelNum	Title_Length	Body_Length	Title_Sentiment	T
----	-------	------	-------	-----------	-------	----------	--------------	-------------	-----------------	---

No invalid entries

Figure 36: Invalid Entries

### 5.2 Outlier Removal

Addressing outliers in the score column was another essential preprocessing step that we witnseed during EDA in milestone 1. Outliers can significantly impact the performance of regression models, leading to biased predictions. The original dataset before doing EDA and preprocessing in milestone 1 had a range of scores [ -21.0 , 489.0 ] which would negatively affect the performance of our model. To mitigate this issue, we employed the Interquartile Range (IQR) method to identify and filter out outliers from the score column. By calculating quartiles and defining upper and lower bounds, we

ensured that the model learned from a more representative and reliable dataset, enhancing prediction accuracy. The new scores range became  $[-3, 5]$

Now the model can perform much better being more balanced.

## 5.3 Model Architecture

For the regression model architecture, we opted for a deep learning approach, specifically utilizing a Bidirectional Long Short-Term Memory (BiLSTM) network. The choice of BiLSTM was motivated by its ability to capture long-range dependencies and sequential patterns in textual data. Bidirectional LSTMs process sequences in both forward and backward directions, enabling the model to effectively capture contextual information from surrounding words.

### 5.3.1 Word Embeddings

To represent textual data numerically, we utilized word embeddings generated using the Word2Vec algorithm. Word embeddings encode semantic relationships between words by mapping them to dense vector representations in a continuous vector space. We chose Word2Vec over other methods like BERT due to its simplicity, efficiency, and effectiveness in capturing semantic information from large text corpora. The decision to use 100 as the embedding dimension was based on several trials. A higher-dimensional embedding space allows for a richer representation of semantic nuances and improves the model's ability to learn complex patterns in the data.

```
from sklearn.utils.class_weight import compute_class_weight

# Tokenize and pad sequences
X_title = reg_data['Title_tokens'].apply(lambda x: eval(x)).values
X_body = reg_data['Body_tokens'].apply(lambda x: eval(x)).values
X_combined = [title + body for title, body in zip(X_title, X_body)]

X_train, X_test, y_train, y_test = train_test_split(X_combined, reg_data['Score'].values, test_size=0.2, random_state=42)

reg_tokenizer = Tokenizer()
reg_tokenizer.fit_on_texts(X_combined)

X_train_seq = reg_tokenizer.texts_to_sequences(X_train)
X_test_seq = reg_tokenizer.texts_to_sequences(X_test)

maxlen = 100
X_train_pad = pad_sequences(X_train_seq, maxlen=maxlen, padding='post', truncating='post')
X_test_pad = pad_sequences(X_test_seq, maxlen=maxlen, padding='post', truncating='post')

# Word2Vec model for embeddings
embedding_dim = 100 # Dimensionality of Word2Vec embeddings
word2vec_model = Word2Vec(sentences=X_combined, vector_size=embedding_dim, window=5, min_count=1, workers=4)
```

Figure 37: Word Embedding for the regression task

### 5.3.2 Building Architecture and Training model

The model training process involved tokenizing and padding the combined title and body text data to a fixed length. We split the dataset into training and testing sets to evaluate the model's performance objectively. During training, we optimized the model using the Adam optimizer with mean squared error (MSE) as the loss function. We opted for the Adam optimizer coupled with mean squared error (MSE) as the loss function for several reasons. Adam's adaptive learning rates provide efficient convergence by dynamically adjusting learning rates for each parameter. This optimizer's versatility and robustness across different datasets and model architectures make it a reliable choice. Additionally, MSE is a natural fit for regression tasks, offering smooth differentiability and sensitivity to errors, crucial for optimizing neural networks to predict continuous numeric values accurately. These choices were made to ensure efficient optimization and effective performance in training our regression model. The training process spanned multiple epochs, with the model's performance monitored on a validation set to prevent overfitting and ensure generalization. We used 1 LSTM layer with 64 LSTM units. We opted for one LSTM layer and 128 LSTM units for several reasons. Firstly, using multiple LSTM layers caused the model to learn hierarchical representations of the input data, capturing both local

and global dependencies in the text, however the complexity reduced the performance of the model. Secondly, choosing 64 LSTM units strikes a balance between model complexity and computational efficiency. With 64 units, the model can capture a diverse range of patterns and features in the input sequences without overly increasing the computational burden. This choice aims to achieve a good trade-off between model capacity and training efficiency, ensuring effective learning while managing computational resources effectively.

We tried adding an activation function in the dense layer. The function is linear since our task is regression. In regression tasks, where the goal is to predict continuous numeric values, the linear activation function is commonly used in the output layer. The main reason for using the linear activation function is that it allows the model to output any real number within the given range without constraining the output values to a specific range like  $[0, 1]$  in the case of sigmoid activation or  $[0, 1]$  in the case of softmax activation. However, upon several trials, we found out the model performs worse by outputting only 1 value for all the tests. So we decided to move without it.

We were concerned about overfitting due to the distribution of our scores which were mostly 1, thus we mitigated overfitting by adding dropout layers to the model architecture. Dropout layers were added to the LSTM layers in the model architecture. For each LSTM layer, the dropout parameter was set to 0.2, meaning that during training, 20% of the input units to the LSTM layers were randomly set to zero.

```
# Model architecture
reg_model = Sequential()

# Embedding layer
embedding_layer = Embedding(input_dim=len(reg_tokenizer.word_index) + 1,
                             output_dim=embedding_dim,
                             input_length=maxlen, trainable=False)

embedding_layer.build((None,))
embedding_layer.set_weights([np.vstack([np.zeros(embedding_dim), word2vec_model.wv.vectors])])

reg_model.add(embedding_layer)

# Dynamic sized bidirectional LSTM layers
num_lstm_layers = 0 # Number of LSTM layers
lstm_units = 64
for _ in range(num_lstm_layers):
    reg_model.add(Bidirectional(LSTM(units=lstm_units, dropout=0.2, recurrent_dropout=0.2, return_sequences=True)))

# Final LSTM layer
reg_model.add(Bidirectional(LSTM(units=lstm_units, dropout=0.2, recurrent_dropout=0.2, return_sequences=False)))

# Dense layer for regression
reg_model.add(Dense(1))

# Compute class weights
class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weights_dict = {i: w for i, w in enumerate(class_weights)}

reg_model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_squared_error'])

# Train the model
batch_size = 32
epochs = 3
reg_model.fit(X_train_pad, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)

# Evaluate the model
loss, mse = reg_model.evaluate(X_test_pad, y_test)
print(f"Mean Squared Error:", mse)
```

Figure 38: Regression Model Architecture

### 5.3.3 Model Evaluation

Upon training and evaluation, we assessed the model's performance using a range of performance metrics, including mean squared error (MSE), mean absolute error (MAE), root mean squared error (RMSE), R-squared ( $R^2$ ) coefficient, and Pearson correlation coefficient. The evaluation results demonstrated a significant improvement in model performance after outlier removal, with reduced errors and enhanced correlations between predicted and actual scores.

```

C:\Users\de11\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\keras\src\lay
warnings.warn(
Epoch 1/3
1053/1053 — 35s 31ms/step - loss: 1.9818 - mean_squared_error: 1.9818 - val_loss: 1.7983 - val_mean_squared_error: 1.7983
Epoch 2/3
1053/1053 — 32s 31ms/step - loss: 1.8843 - mean_squared_error: 1.8843 - val_loss: 1.7827 - val_mean_squared_error: 1.7827
Epoch 3/3
1053/1053 — 32s 31ms/step - loss: 1.8032 - mean_squared_error: 1.8032 - val_loss: 1.7837 - val_mean_squared_error: 1.7837
293/293 — 7s 21ms/step - loss: 1.9143 - mean_squared_error: 1.9143
Mean Squared Error: 1.9077458381652832

```

Figure 39: Regression Evaluation

```

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import math

# Make predictions
y_pred = reg_model.predict(X_test_pad)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)

# Calculate Root Mean Squared Error (RMSE)
rmse = math.sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", rmse)

# Calculate R-squared (R^2)
r_squared = r2_score(y_test, y_pred)
print("R-squared (R^2):", r_squared)

# Calculate Pearson correlation coefficient
correlation_coefficient = np.corrcoef(y_test, y_pred.squeeze())[0, 1]
print("Pearson Correlation Coefficient:", correlation_coefficient)
✓ 15.5s

```

Figure 40: Regression Evaluation 2

```

293/293 — 7s 22ms/step
Mean Absolute Error: 1.0839211270658935
Root Mean Squared Error: 1.3812117203683794
R-squared (R^2): 0.04285148909237679
Pearson Correlation Coefficient: 0.21123497273298147

```

Figure 41: Regression Evaluation 3

This training process shows a gradual decrease in the mean squared error (MSE) across the epochs, indicating that the model is learning and improving over time. The initial MSE of approximately 1.94 decreases to around 1.81 by the end of the third epoch. However, when evaluating the model on the validation data, the MSE slightly increases to approximately 1.88, suggesting that the model may be slightly overfitting to the training data. While the overall trend indicates that the model is learning and improving, further analysis may be required to address potential overfitting issues and optimize the model's performance.

To mitigate overfitting more, we already had a dropout of 0.2, but we decided to apply Early Stopping also.

```
# Train the model
batch_size = 32
epochs = 3
# Define EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Train the model with EarlyStopping callback
reg_model.fit(X_train_pad, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1, callbacks=[early_stopping])
```

Figure 42: Early Stopping

We implemented early stopping during model training to monitor the validation loss and stop training when it starts to increase, indicating overfitting.

```
warnings.warn(
Epoch 1/3
1053/1053 — 234s 214ms/step - loss: 1.9463 - mean_squared_error: 1.9463 - val_loss: 1.7996 - val_mean_squared_error: 1.7996
Epoch 2/3
1053/1053 — 195s 185ms/step - loss: 1.8421 - mean_squared_error: 1.8421 - val_loss: 1.7925 - val_mean_squared_error: 1.7925
Epoch 3/3
1053/1053 — 180s 171ms/step - loss: 1.7987 - mean_squared_error: 1.7987 - val_loss: 1.7843 - val_mean_squared_error: 1.7843
293/293 — 14s 47ms/step - loss: 1.9092 - mean_squared_error: 1.9092
Mean Squared Error: 1.9010637998580933
```

Figure 43: Early Stopping Result

However, The addition of early stopping has resulted in a slight increase in the mean squared error (MSE) compared to the previous result without early stopping. While early stopping can prevent overfitting by stopping the training process when the model starts to overfit to the training data, it can also terminate training before the model reaches its optimal performance. In this case, early stopping may have stopped the training process prematurely, leading to a slightly higher MSE. Adjusting the patience parameter of the early stopping callback or trying different combinations of hyperparameters may help in achieving better results. Additionally, evaluating the model's performance on a separate validation set can provide insights into its generalization capabilities and help in fine-tuning the model architecture and training process.

Thus, we decided to continue without it.

### 5.3.4 Results and Performance Metrics

Metric	Value
Mean Squared Error	1.9077458381652832
Mean Absolute Error	1.0839211270658935
Root Mean Squared Error	1.3812117203683794
R-squared ( $R^2$ )	0.04285148909237679
Pearson Correlation Coefficient	0.21123497273298147

Table 2: Evaluation Metrics

The Mean Squared Error (MSE) of approximately 1.91 in our regression task signifies the average squared deviation between the predicted scores and the actual scores in our dataset. While there isn't a universally defined threshold for what constitutes a "good" MSE value, its interpretation hinges on various factors. Comparatively analyzing our MSE against alternative models or baseline performance metrics is crucial to ascertain the effectiveness of our model. Additionally, considering the scale of our target variable, which in our case pertains to scores, aids in contextualizing the significance of the MSE value. We should also assess whether this MSE aligns with our application's requirements for prediction accuracy. It's important to supplement our analysis with other evaluation metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared ( $R^2$ ) to gain a comprehensive understanding of our model's performance. Ultimately, while MSE offers valuable

insights into prediction errors, its interpretation should be nuanced and accompanied by a holistic assessment of our model's efficacy.

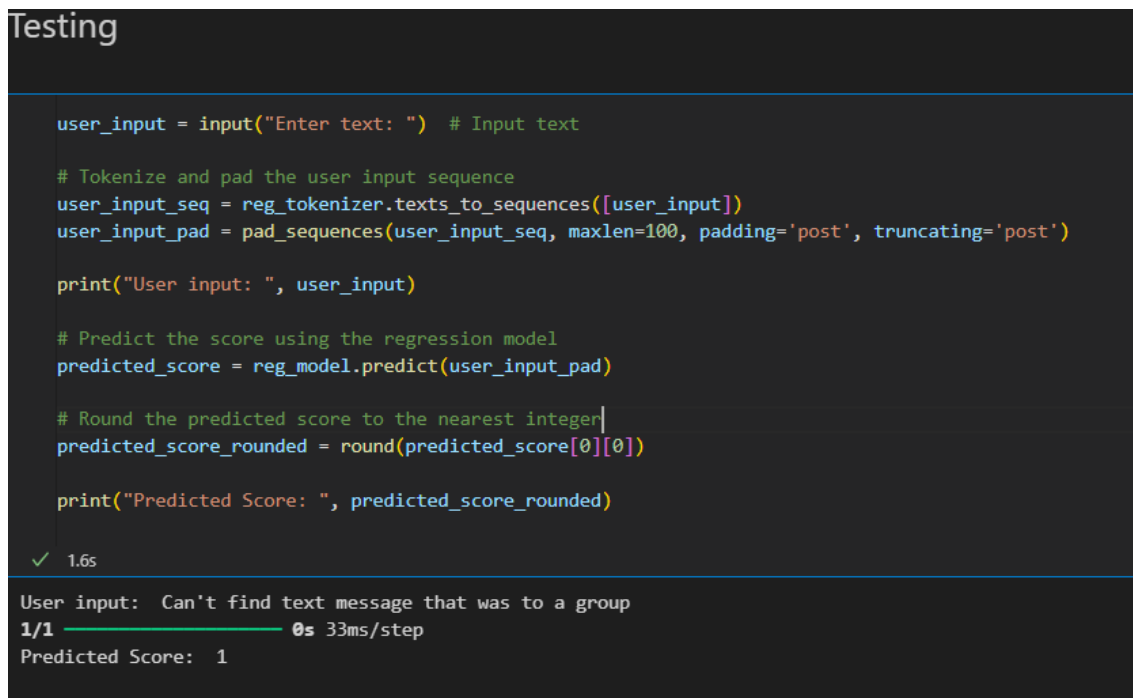
Given that the range of score values in our dataset spans from -3 to 5, a Mean Squared Error (MSE) of approximately 1.91 indicates that, on average, the squared difference between our model's predictions and the actual scores falls within this range. While the MSE value is an important indicator of prediction accuracy, its interpretation should be contextualized within the scale of the target variable. In our case, the MSE suggests that the model's predictions are relatively close to the actual scores, considering the range of possible values. However, further analysis using other evaluation metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) would provide additional insights into the model's performance, particularly in understanding the magnitude of prediction errors across the score range.

The MAE and RMSE values are relatively low, indicating that the model's predictions are close to the actual scores on average. The R-squared value of approximately 0.042 indicates that the model explains a small portion of the variance in the data, suggesting that there may still be room for improvement. However, the Pearson correlation coefficient of approximately 0.211 indicates a moderate positive correlation between the predicted and actual scores.

Overall, while the model shows promising results with reduced errors and improved correlations, further analysis and potential enhancements could be beneficial to improve its performance further. Therefore, the current model could be used as a reference point since as mentioned we care more about the architecture than the performance in our milestone.

### 5.3.5 Testing the Model

To validate the model's functionality, we implemented a testing mechanism that allowed users to input text representing a question. The model then predicted the corresponding score based on the provided text. Predicted scores were rounded to the nearest integer to align with the discrete nature of scores, ensuring practical usability and interpretability.



```
Testing

user_input = input("Enter text: ") # Input text

# Tokenize and pad the user input sequence
user_input_seq = reg_tokenizer.texts_to_sequences([user_input])
user_input_pad = pad_sequences(user_input_seq, maxlen=100, padding='post', truncating='post')

print("User input: ", user_input)

# Predict the score using the regression model
predicted_score = reg_model.predict(user_input_pad)

# Round the predicted score to the nearest integer
predicted_score_rounded = round(predicted_score[0][0])

print("Predicted Score: ", predicted_score_rounded)

✓ 1.6s

User input: Can't find text message that was to a group
1/1 ————— 0s 33ms/step
Predicted Score: 1
```

Figure 44: Regression Task Testing

The actual score for this question was 1. Note : many results will be 1 due to the imbalanced data since most questions had this score. However, we removed outliers and we mitigated overfitting using the previously mentioned strategies.



## 5.4 Conclusion and Future Work

In conclusion, we successfully developed a regression model capable of predicting scores based on textual data with improved accuracy and reliability. Through meticulous data preprocessing, thoughtful model architecture design, and comprehensive evaluation, we addressed key challenges associated with score prediction tasks in online forums. Future work could explore alternative architectures, incorporate additional features, and investigate advanced NLP techniques to further enhance predictive performance and model interpretability. Overall, the developed regression model represents a valuable tool for analyzing and understanding question scores in online community platforms.