**Sentiment Classification**

**Introduction**

This task is related to Sentiment analysis of a dataset named FiQA which consists of English news headlines and social media post related to finance. Each headline or social media post has a label which shows the sentiment of the respective text. There are three classes in the whole dataset, (0) for negative, (1) for neutral and (2) for positive. The dataset can be downloaded from the following website: <https://sites.google.com/view/fiqa/home>. Our task is to build a machine learning model that will classify the news headline or social media post into one of three classes based on the text.

Briefly explain how your chosen method works and its main strengths and limitations

**Methods**

In this work, we used two machine learning algorithms to classify the news headlines and social media post.

* Multinomial Naive Bayes
* Logistic Regression

**Naive Bayes**: Naive Bayes is a machine learning algorithm that is used for classification problems. It is based on Bayes' theorem, which is a mathematical formula that describes the probability of an event occurring, based on prior knowledge of conditions that might be related to the event. Multinomial Naive Bayes (MNB) is a variant of the Naive Bayes algorithm that is particularly suited for text classification tasks. Multinomial Naive Bayes assumes that the features in the input data are generated from a multinomial distribution, which makes it well-suited for problems where the input data is represented as counts of occurrences of various features. Multinomial Naive Bayes is simple and fast, making it well-suited for large-scale text classification tasks. Multinomial Naive Bayes is robust to irrelevant features, which means that it can still perform well even when there are many irrelevant or noisy features in the input data however Multinomial Naive Bayes assumes that the features (i.e., word frequencies) are independent, which is not always true in practice. This can limit the accuracy of the algorithm, especially when there are strong correlations between features.

**Logistic Regression:** Logistic regression is a statistical model used for binary classification problems, where the goal is to predict a binary output (such as yes/no or true/false) based on one or more input variables. The logistic regression model uses a logistic function to model the probability of the output variable being true, given the input variables. Logistic regression can handle both numerical and categorical input variables and can be extended to handle multi-class classification problems. It is a linear model, which makes it less prone to overfitting compared to more complex models and can perform well in situations where the decision boundary is approximately linear however Logistic regression assumes that the relationship between the input variables and the output variable is linear. This may not be the case in real-world applications and can limit the accuracy of the algorithm.

Describe the preprocessing steps and the features you use to represent each text instance.

**Preprocessing steps**

In this work, we used some preprocessing steps to clean the text data.

* **Lowercase conversion**: In this step, we converted text data into lowercase.
* **Removal of links:** In this step, we removed web links from text data using “re” library.
* **Removal of digits and symbols**: In this step, we removed digits and symbols from text.
* **Removal of stopwords:** In this step, we removed English stopwords.
* **Lemmatization:** In this step, we lemmatized the words and convert each word into its root.

After cleaning the text data, we used CountVectorizer() class of Sci-kit learn library used to convert a collection of text documents into a matrix of token counts. It is a technique for vectorizing text data in which a text document is represented as a vector of word counts. we took these vectors as features of text data.

Explain why you chose those features and preprocessing steps and hypothesise how they will  
affect your results

The use of CountVectorizer() allows us to represent text data as numerical data that can be used in machine learning algorithms. By counting the occurrences of words in a text, we can identify which words are most important for determining the meaning or sentiment of the text. This allows us to create features for each text document that can be used as input to a machine learning model.

Briefly describe your software implementation

**Implementation**

In this work, we used two machine learning algorithms MultinomialNB and LogisticRegression from “Sci-kit learn” library. First, we trained a Multinomial Naive Bayes (NB) classifier using the fit method on the training data x\_train and corresponding training labels train\_labels. The MultinomialNB() function initializes an instance of the Multinomial Naive Bayes classifier with default hyper parameters. The default parameters of MultinomialNB() are alpha, fit\_prior and class\_prior. Alpha is a smoothing parameter that prevents zero probabilities in the model. The default value is 1.0, which represents Laplace smoothing. Fit\_prior is a Boolean parameter that indicates whether or not to learn the class prior probabilities from the training data. The default value is True, which means that the prior probabilities are learned from the data. Class\_prior is an optional parameter that allows the user to specify the prior probabilities for each class. If none, the prior probabilities are set based on the fit\_prior parameter. Then we trained a logistic regression classifier using the fit method on the training data x\_train and corresponding training labels train\_labels. The most important parameter of LogisticRegression is penalty: This is a regularization parameter that is used to prevent over fitting. The default value is l2, which represents L2 regularization. After training, we predicted the results using test split and compared it with original labels. The accuracy of Naïve bayes and Logistic Regression on test data are 66% and 65% respectively.

Define your performance metrics and state their limitations.

**Performance metrices**

We used four performance metrices for this classification task.

* **Accuracy:** It measures the percentage of correctly classified instances out of the total number of instances in the dataset.
* **Precision:** It measures the proportion of true positives (instances correctly predicted as positive) out of the total number of instances that the model predicted as positive.
* **Recall:** It measures the proportion of true positives (instances correctly predicted as positive) out of the total number of actual positive instances in the dataset.
* **F1-score:** It is the harmonic mean of precision and recall and provides a balance between the two metrics.

Precision and recall are sensitive to the imbalance between the classes in the dataset however F1 score provides a balance between precision and recall, but it may not always be the best metric to use depending on the specific problem and the importance of precision versus recall. Accuracy can also be misleading in the presence of class imbalance.

Describe the testing procedure (e.g., how you used each split of the dataset

**Testing procedure**

After training the models, we tested the performance of the model on test data using predict() function. After getting the result on test data, we visualized it in the form of bar graph. We used train split containing 512 samples to train the models and test split which also consists of 512 samples to check the performance of the model

Chart, bar chart

Description automatically generatedShow your results using suitable plots or tables.

Can you identify common themes or topics associated with negative sentiment or positive  
sentiment in this dataset?

Yes, we can common themes or topics associated with negative sentiment or positive  
sentiment in this dataset

Explain the method you use to identify themes or topics.

**Topics Extraction**

A word cloud is a visualization technique that displays a collection of words in a way that conveys the relative frequency or importance of each word. In this work, we used “wordcloud” library of python language to extract the topics from the text data. We used WordCloud() object to generate a word cloud of the text data by giving it specific parameters.

Show your results (e.g., by listing or visualizing example topics or themes)



Interpret the results and summarize the limitations of your approach.

In a word cloud, words are arranged randomly in a cluster or cloud, with the size and/or color of each word representing its frequency or importance. Typically, more frequent or important words are displayed larger and/or in bolder text.

**Name Entity Recognition**

**Introduction**

This task is related to Name Entity Recognition of a dataset which consists of journal articles from fields including medicine and pharmacy. The data is annotated with five entity types: DNA, protein, cell type, cell line, RNA. The dataset can be downloaded from the following link: <https://huggingface.co/datasets/tner/bionlp2004>. Out task is to build a tool that will recognize the medical entities in text data and will provide us the label of that entity.

Explain how your chosen method works and its main strengths and limitations

**Methods**

We used a popular NER model named “en\_core\_web\_sm” of spacy 2.3.9 library which can be used for various natural language processing tasks such as tokenization, part-of-speech tagging, named entity recognition, and dependency parsing. It is also compatible with spacy and is specifically designed to work with the spacy library, which is widely used for NLP tasks. This model is a small model, which makes it faster to load and use than larger models and provides good accuracy but due to its small size, the en\_core\_web\_sm model may not be able to handle certain types of text or tasks that require more specialized language knowledge. This model performs well on common named entity types such as PERSON, ORG, and GPE, it may not perform as well on more specific or rare named entity types.

Briefly explain how entity spans are encoded as tags for each token in a text.

**Text Encoding**

We wrote a Python script that generates training data for named entity recognition (NER) from raw text and corresponding entity labels. The script takes in two input lists: train\_sentences, which contains a list of raw text sentences, and train\_labels\_ner, which contains a list of corresponding entity labels for each sentence. The output is a list of tuples, where each tuple contains a sentence and a dictionary of entity annotations for that sentence. The code iterates over each sentence and corresponding entity label in the input lists. For each sentence, it creates an empty list entities\_list to hold the entity annotations for that sentence, as well as an empty dictionary tags\_dict to hold the entity annotations in a format compatible with spacy. For each word in the sentence and its corresponding label, the script checks if the label is not '0', which indicates that the word is not a part of any named entity. If the label is not '0', the script extracts the start and end indices of the word in the sentence and maps the label to its corresponding key using a dictionary id2label. The script then creates an entity annotation tuple containing the start index, end index, and label key, and appends this tuple to the entities\_list. After processing all words in the sentence, the script creates a tuple containing the sentence and a dictionary with a single key 'entities', whose value is the list of entity annotations for that sentence. This tuple is appended to the train\_data list.

Briefly describe your software implementation.

**Implementation**

We defined a function called train\_model that takes train\_data and iterations. train\_data is a list of tuples, where each tuple contains a sentence and a dictionary of entity annotations for that sentence. Iterations is an integer that specifies the number of iterations for train the model. The function first loads the pre-trained en\_core\_web\_sm model using the spacy.load function. It creates a new “ner” pipeline component and adds it to the model.The function then iterates over each sentence in the train\_data and adds the entity labels to the ner pipeline component using the ner.add\_label method.It then initializes an optimizer using the nlp.begin\_training method.The function then trains the model using a loop that iterates iterations times. For each iteration, the training data is shuffled using the random.shuffle method. The training data is then divided into batches using the minibatch function, which takes in the train\_data list and two other arguments that specify the minimum and maximum batch sizes. The minibatch function returns a generator that yields batches of training data. For each batch of training data, the function extracts the sentences and annotations using the zip function. It then updates the ner pipeline component using the nlp.update method, which takes in the sentences, annotations, optimizer, dropout rate, and a dictionary to store the losses. After training the model for the specified number of iterations, the function returns the trained model.

Detail the features you have chosen, why you chose them, and hypothesis how your choice.  
will affect your results.

we chose a list as feature in which there is a text string and a tuple containing the starting and ending index of the name entity in that text. As we have rare entities, so this model did not perform well on this dataset.

Explain your choice of performance metrics and their limitations.

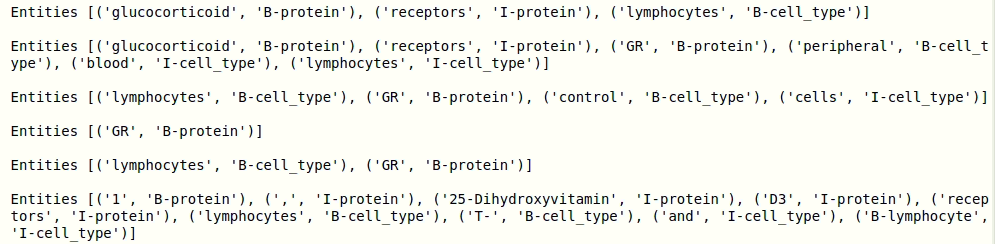
**Evaluation**

We used nlp.update method that updates the model parameters based on the current batch of training data. Specifically, it computes the gradients of the loss with respect to the model parameters using back propagation, and then updates the parameters using the optimizer. The training loss is also computed and added to the losses dictionary. After updating the model, the code prints the value of the losses dictionary for the current batch. This allows us to monitor the progress of the training and check if the loss is decreasing over time, which is an indication that the model is improving.

**Testing**

After training, we saved the model by using model.to\_disk() method and loaded the model using spacy.load() Then, we looped through a list of test sentences and uses the loaded model to predict the named entities in each sentence. For each sentence, the model object is called with the current test\_sample as its argument, which returns a Doc object representing the parsed document. Then, the doc.ents attribute is used to extract the named entities found by the model. The named entities are printed as a list of tuples containing the entity text and its predicted label.

Show your results using suitable plots and/or tables.



How could you improve the method or experimental process? Consider the errors your  
method makes.

The loss rate at the training time is high because this model works better for comman entities but we have rare medical entities.