Final Report

Introduction/Background

Text emotion detection is one of the challenging problems in NLP that involves classifying the emotional content of written text. With the increasing use of applications involving communication such as social media, messaging websites, and other human-computing applications, it is essential for applications to accurately understand the emotional tone of textual content.

For this project, we will use Go Emotions datasets from Google, which consists of Reddit comments labeled with 27 different emotion categories. This data set provides a rich and diverse set of emotional expressions, enabling the model to learn and classify emotions effectively.

Problem Definition

Unlike traditional sentiment analysis that categorizes text into simple positive, negative, or neutral sentiments, our project aims to classify a much more detailed range of emotions. As shown by the image below, there's dozens of emotions that may be similar, but have their own unique meaning that needs to be recognized in sentences.

As mentioned previously, prior research has looked into detecting emotions in sentences, but have only classified them into broad categories. For instance, a study done in 2021 successfully classified emotions from

texts, but only grouped them in 4 broad categories, as shown below [1]:

Class	Actual	% each class	Sampling	% changes	Train	Test
Anger	1701	24%	1701	0%	1446	255
Sadness	1533	22%	1700	11%	1445	255
Fear	2252	32%	1700	-25%	1445	255
Joy	1616	23%	1700	5%	1445	255
TOTAL	**7102**	**100%**	**6801**	**-4%**	**5781**	**1020**

A second research study done in 2022 was also successful in detecting emotions in sentences, but used data that had “173 emotions but grouped into 7 emotions” [2]. Another previous research study attempted to detect more specific emotions but had difficulty. The researchers were unsuccessful in detecting the difference between negative sentiments such as “anger, sadness, or fear” in their algorithms [3].

Our project aims to classify a wide range of emotions in text. By developing a robust text emotion detection system, we can bridge the gap between simple sentiment analysis and humans' complex emotions, leading to more meaningful and engaging interactions between users and machines.

Methods

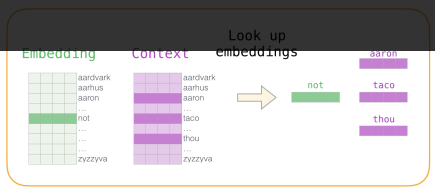
Regarding machine learning algorithms, we will use logistic regression, neural network and transformer models. Logistic regression offers a fast and simple classification approach, representing a good baseline model. The NN model could be used to quickly analyze and classify a large amount of data, which is often used in other NLP models. Transformers are state-of-the-art models that leverage attention mechanisms and excel in handling complex relationships in sequential data.

1. Logistic regression

Data processing: word2vector

In natural language processing (NLP), transforming textual data into a numerical form is crucial for feeding it into machine learning models. Two popular techniques for this are Word2Vec and Bag of Words (BoW). Each has its strengths, but Word2Vec is often preferred for capturing semantic relationships and producing efficient word representations.

Why Word2Vec?



a. Semantic Understanding: Unlike BoW, which relies on word frequency and treats words independently, Word2Vec captures the context and meaning of words by considering their relationships with neighboring words. This allows it to encode semantic similarities, enabling better handling of synonyms and related terms.

b. Efficient Representations: Word2Vec generates low-dimensional word vectors, making embeddings computationally efficient. These smaller representations reduce memory usage compared to the high-dimensional vectors produced by BoW, especially in large corpora.

c. Improved Synonym Handling: Words with similar meanings or contexts are mapped to close points in the vector space, which enhances the model's ability to generalize across different but related words.

ML model: Logistic regression

Logistic regression is a fundamental machine learning algorithm that is widely used for classification tasks. It uses a linear equation to compute a weighted sum of input features, adds a bias term, and then applies a sigmoid (logistic) function to map the result to a value between 0 and 1. This output probability is used to classify data points, typically by applying a threshold (e.g., 0.5) to decide the class. Logistic regression is ideal for binary classification but can be extended to multiclass problems using techniques like one-vs-rest or softmax.Despite its simplicity, it is highly effective and offers several advantages that make it a popular choice, especially for projects requiring quick implementation and clarity.

Why Logistic Regression?

Logistic regression is easy to set up and requires minimal tuning compared to more complex machine learning models. Key features include: 1. Straightforward Algorithm: It works by estimating probabilities using a

logistic (sigmoid) function, which maps predicted values to a range between 0 and 1. This makes it intuitive for binary classification tasks, such as predicting whether an email is spam or not. 2. Feature Coefficients: Logistic regression provides coefficients for each feature, which can be interpreted as the impact of that feature on the outcome. This transparency is valuable in domains like healthcare, finance, and social sciences, where understanding the decision-making process is critical.

Also, one of the standout features of logistic regression is its speed: 1. Low Computational Overhead: The algorithm involves simple matrix operations and does not require iterative computations like deep learning models, making it computationally lightweight. 2. Efficiency on Small to Medium-Sized Data: Logistic regression excels on datasets that are not excessively large, making it suitable for scenarios where quick results are needed without requiring significant computing resources. 3. Scalability: While it is most efficient on smaller datasets, logistic regression can scale reasonably well with optimization techniques like stochastic gradient descent (SGD), allowing it to handle larger datasets if needed.

2. Neural Network

Data processing: Pre-trained Transformer

First of all, the Goemotion dataset we choose contains 58k comments extracted from Reddit, labeled with a subset of 28 different emotions (mostly labeled with only one emotion, but multi-label cases exist). It has 37 columns, one holding the comments as a string, eight holding source and rater information of each comment, and 28 holding emotion labels in a manner similar to one-hot. The texts column, which is to be our raw input data, features a frequent presence of rare words like internet slang and abbreviation (for example, "kinda", "Seeeee!", "Remindme") which make it difficult to tokenize using simple method like bag-of-word. These properties of the dataset require us to remove irrelevant columns and to combat rare words with tokenizer that is pre-trained with social media comments.

On the other hand, to achieve our goal of sentiment analysis, we should consider the relationship between text and the embedded emotion in general. Emotion, as yet an additional layer above the mere semantics of a sentence, finds it dwellings not only in the content, but also in the form of a sentence, that which could normally be neglected if we are only focusing on the semantics. This form would include syntax, morphology, voice, modal particle and punctuations, etc. This would forbid us from using tokenization optimization that reduces words to stem, which whips out morphology, but instead promotes sub-word tokenization that would keep some morphological structures. Also, this would drive us away from embedding methods that whips out grammatical information entirely, like bag-of-word/TF-IDF which counts nothing more than frequency of words, or context-free word embeddings which takes into account nothing but the semantics. As a result, the embedding method that is left is contextualized embedding.

According to these analysis, we implemented the following data processing methods: For data cleaning, we removed the eight columns of irrelevant information, and also the rows where the column ”example_very_unclear” are marked with “True”, since through some manual inspection we determined that they are too controversial and might confuse our model. We then imported a BERT-based transformer model called “bertweet” that is pre-trained with comments extracted from tweeter, which provide both tokenizer and contextualized embedding. This model can deal with the rare word problem of the dataset effectively. We then took the data after tokenization and embedding (the embedding of token [CLS] which is seen as the embedding for the whole sentence rather than any single word) as our processed data, on which our ML model is ready to fit. The resultant dataset had 207814 rows and 768 columns, which is a BERT standard number. We tried conducting PCA which could reduce the number of features to around 300, 200, and 130 if preserving variance of 98, 95, 90, respectively, yet whether we could cut off directly from the BERT output without disrupting the proper function of the embedding is still unclear. All the processed dataset can be found in the “\source\data” folder.

ML model: NN

A neural network is a machine learning model inspired by the human brain, designed to recognize patterns and solve complex problems. It consists of layers of interconnected nodes: the input layer processes data, hidden layers transform it through weighted connections and activation functions, and the output layer generates predictions. During training, the network learns by comparing its predictions to true values, calculating errors, and adjusting weights and biases through a process called backpropagation. This iterative process allows the network to improve and perform tasks like classification, prediction, or image recognition.

A NN model could be used to quickly analyze and classify a large amount of data quickly, which is often used in other NLP models. We chose the simple neural network instead of the convolutional neural network as it is more suitable for sentiment analysis, and also, we chose it over the transformer like the one we used for data processing since it's less computationally intensive.

3. Transformer(BERT)

ML model

The Transformer, above the usual neural network architecture, relies heavily on the self-attention mechanism, allowing models to weigh the importance of each word in a sequence concerning others. This approach overcomes the limitations of previous recurrent and convolutional architectures by enabling parallelization and capturing long-range dependencies effectively.

BERT(Bidirectional Encoder Representations from Transformers), introduced by Google in 2018, is a landmark model in NLP. Unlike earlier models that processed text unidirectionally (left-to-right or right-to-left), BERT uses a bidirectional approach. This enables it to consider both previous and subsequent context simultaneously, making it highly effective for tasks requiring deep semantic understanding. The pre-trained tokenizer used in our neural network model is a variant of BERT.

Since each BERT has millions of parameters, a more convenient and common way to utilize BERT is to fine-

tune a pre-trained BERT model on specific down-stream tasks, emotion detection in our case. We've formulated our code for fine-tuning with many BERT-related libraries, including Hugging Face Fransformer, Simple Transformers, AllenNLP and TweetNLP, but many have failed due to various difficulties, mainly environment issues and dependency conflicts. Eventually we chose flair, a state-of-the-art and easy-to-use NLP library. Also, we came across a pre-trained BERT model, provided by pysentimiento, trained also on the Goemotion dataset, but has compressed its prediction labels to the more commonly used 6 basic emotions [4]. So in addition to fine-tuning a BERT model over the Goemotion dataset, we've explored the relationship between the six basic emotions and the more fine-grained emotion taxonomy used by Goemotion by attempting to fit a NN model between the pysentimiento's outcome and Goemotion's labels.

Results and Discussion

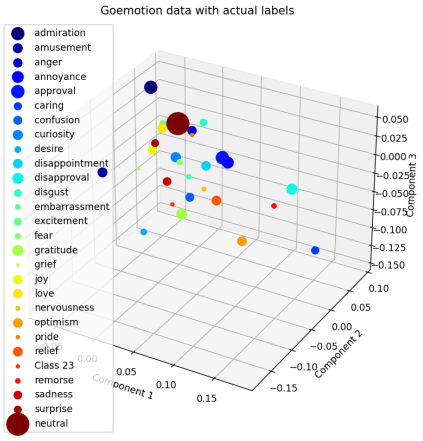


Figure 1: Visualization of Orginial Dataset

1. Logistic regression

Results

With the test_size set to be 0.3, max_iter=5000, we achieved our best performance with accuracy 30.80%.

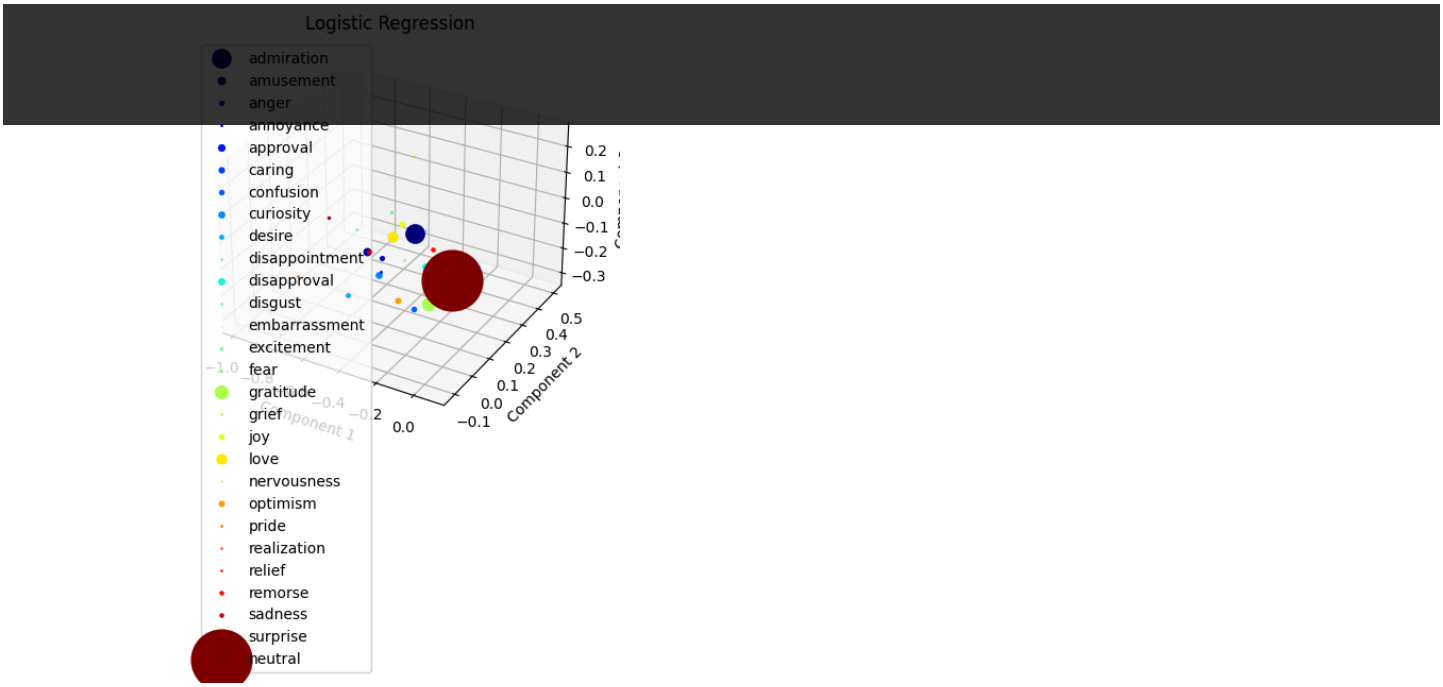


Figure 2: Visualization of Predicted Result (Logistic Regression)

Analysis

Changing the test_size does not affect the final accuracy too much, in most cases generates accuracy around 0.3.

Possible cause of low accuracy: lr assumes a linear relationship between features and the outcome, so it may not handle non-linear patterns well enough. (couldn't interpret complex relationships between features) May not be suitable for a very large dataset.

2. Neural Network

Results

We explored two neural network frameworks, which are vanilla PyTorch and PyTorch Lightning.
With vanilla PyTorch, we obtain the best performance of
X_train: Accuracy: 0.3332, F1 Score: 0.3037, Precision: 0.7475, Recall: 0.2318
X_test: Accuracy: 0.2470, F1 Score: 0.2118, Precision: 0.5068, Recall: 0.1644

with three hidden layers, containing 2048, 1024, and 512 neurons respectively, and the following hyperparameters: Batch size: 256, train steps: 50, dropout rate: 0.6, learning rate: 0.0005, weight decay: 1e-06

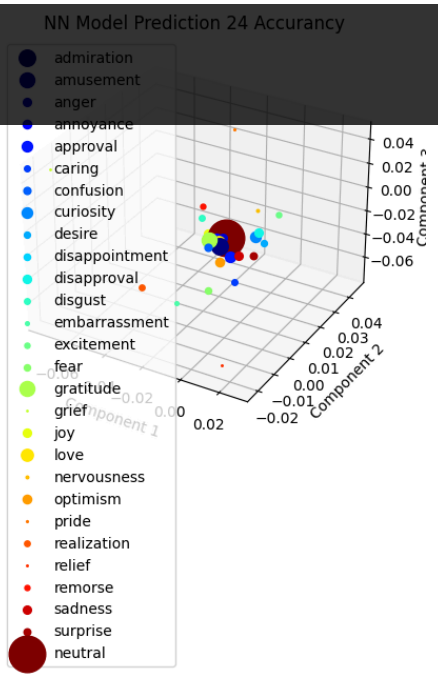


Figure 3: Visualization of Predicted Result (NN-24)

With PyTorch Lightning, we obtain the best performance of
Train: Accuracy: 0.5735, F1 Score: 0.4241
Test: Accuracy: 0.3801, F1 Score: 0.2400

With two hidden layers, each containing 1024 neurons, and the following hyperparameters:
Batch size: 32, train steps: 200, dropout rate: 0.3, learning rate: 0.00008

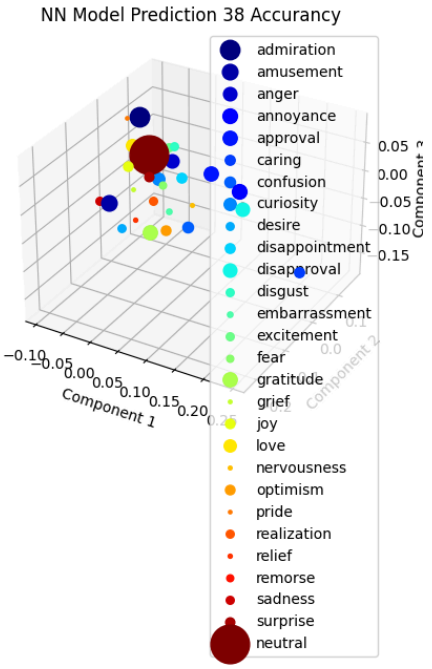


Figure 4: Visualization of Predicted Result (NN-38)

Analysis

Although a tutorial on the dataset's website suggests a training step of 10000, for fine tuning and hyperparameter search purposes, the model is only trained with steps of at most 200. The value of accuracy for the same task on the same dataset is 45.0±0.9 in literature (https://doi.org/10.48550/arXiv.2106.09462), which is not far from our result of 38.0 with Pytorch Lightning. This suggests fitting on such a fine-grained emotion category is in itself a difficult task. Further analysis will reveal part of the reason.

Given that the training accuracy is low, this indicates the model is likely underfitting—it hasn't learned the patterns in the training data well enough to generalize effectively. It is noticed that at least in this range of train steps, nevertheless, increasing the complexity of the model by adding more hidden layers or more neurons per layer does not improve the underfitting issue, but rather results in worsened performance. Nevertheless, since the best score in literature is still less than a half, the reason for this low accuracy might not be the limited number of epochs in training, it could be, instead, that the existing models cannot yet capture the complex and subtle relationships among this fine-grained set of emotions. On the other hand, while underfitting seems to prevail, overfitting still emerges. Regularization at this stage has improved both overfitting and underfitting issues.

The low F1 score implies that both precision and recall are limited. For example, in the case of vanilla Pytorch, Precision (Train: 0.7475, Test: 0.5068) is relatively high compared to recall, indicating that when the model makes positive predictions, it's right more often than not. Recall (Train: 0.2318, Test: 0.1644) is much lower than precision, meaning the model is failing to identify a large portion of actual positive cases. This suggests the model is likely too conservative in predicting positives and may be biased toward the negative class. This issue might relate to the fact that about 26% of the data points are of the “emotion” neutral, making up the single largest emotion category.

It is also easily noted how the loss gets stuck at around 0.1 and never gets down that bound. In addition,

within our range of steps, we find decreasing the number of hidden layers would often bring better performance. With 2 hidden layers of 2048 and 1024 neurons, we achieved train accuracy of around 40% with vanilla PyTorch. The result we have for Pytorch Lightning surely confirms this tendency.

3. Transformer(BERT)

Results

For Flair Transformer, we obtain the result of
Accuracy: 0.1501, F-score 0.2419
With hyperparameters of
Lr =8e-4, mini_batch_size=32, epoch=4

For pysentimiento Transformer, we obtain the result of
Accuracy: 0.3009
And the neural network classifier attached to which with the same hyperparameters as our NN model, except using 128 neurons per hidden layer.

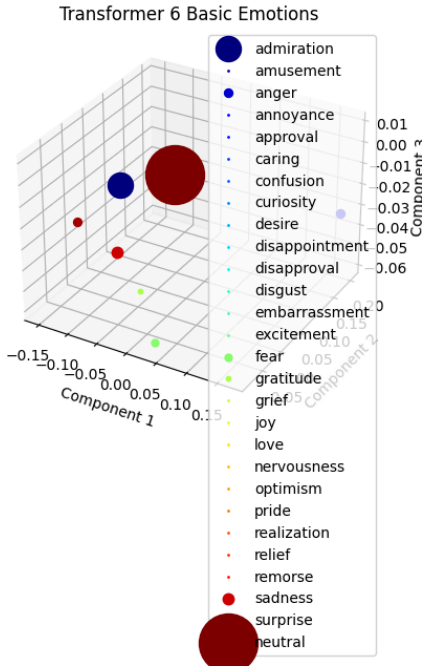


Figure 5: Visualization of Predicted Result (Transformer)

Analysis

Our choice of the flair library and some of its hyperparameters is not that of optimality, especially when it comes to built-in aid for downstream tasks, as the model we use is not specifically for emotion detection and does not specialize in internet slang. The

model does converge, yet the performance doesn't not manifest at all the advantage of the transformer model.

However, the result from pysentimiento provides interesting insights. It is recorded in their paper that the best performance pysentimiento's model has on the 28-label Goemotion dataset is arround 45.0, which is slightly higher than our result of inputting the result from that same model, however, compressed into ekman's 6 emotions, and then applying a NN classifier to restore the 28 labels. Since our NN classifier is deemed to converge as it's score stay around 30 for large modification in hyperparameters, this result suggests that there exists a certain degree of information loss when converting from the 28 emotions to ekman's 6 emotions, which, though natural as it seems, implies that the ekman's 6 emotions are not that basic, since we cannot obtain all of the subtler emotions by merely taking combinations of them. An obvious reason for this lies in the visualization for this result, as the neural class is so great in the number of its member, which one can interpret as the result of data points losing detailed emotional information and getting classified immediately into neutral class, if they do not present strong enough emotional tendency, unlike in the 28-label taxonomy.

	Logistic Regression	Neural Network	Transformer
Strengths	**Simple, Fast, Less prone to overfitting**	**Powerful while cost-effective**	**Powerful for complex, contextual relationships**
Limits	**Limited Expressiveness**	**Interpretability**	**Computationally expensive, hard to customize**
Results & Analysis	Train: Accuracy: 0.5735, F1 Score: 0.4241 Test: Accuracy: 0.3801, F1 Score: 0.2400	Accuracy 0.3080	Flair Transformer: Accuracy: 0.1501 F-score 0.2419 Pysentimiento Transformer: Accuracy: 0.3009

Next Step

With our current models, we'll try to implement the above resolutions to our issues and conduct more extensive hyperparameter search and see if we can obtain better results. We encountered many difficulties trying to fine tune pre-trained transformers, especially for Hugging face Transformers and TweetNLP, which suits our task well. A next step will be to set up these more

suitable libraries to refactor some of our tasks, and to try to achieve performance scores in literature.

References

[1] A. Chowanda, R. Sutoyo, Meiliana, and S. Tanachutiwat, “Exploring Text-based Emotions Recognition Machine Learning Techniques on Social Media Conversation,” Procedia Computer Science, vol. 179, pp. 821-828, 2021, doi: https://doi.org/10.1016/j.procs.2021.01.099.

[2] S. K. Bharti, R. K. Gupta, P. K. Shukla, M. Bouye, S. K. Hingaa, and A. Mahmoud, “Text-Based Emotion Recognition Using Deep Learning Approach,” Computational Intelligence and Neuroscience, vol. 2022, p. 2645381, Aug. 2022, doi: https://doi.org/10.1155/2022/2645381.

[3] Kristína Machová, M. Szabóová, Ján Paralič, and Ján Mičko, “Detection of emotion by text analysis using machine learning,” Frontiers in Psychology, vol. 14, Sep. 2023, doi: https://doi.org/10.3389/fpsyg.2023.1190326.

[4] T. Dalgleish and M. Power, Handbook of Cognition and Emotion. John Wiley & Sons, 2000.

[5] J. M. Pérez et al., “pysentimiento: A Python Toolkit for Opinion Mining and Social NLP tasks,” arXiv.org, Jun. 17, 2021. https://arxiv.org/abs/2106.09462

[6] D. Demszky, D. Movshovitz-Attias, J. Ko, A. Cowen, G. Nemade, and S. Ravi, “GoEmotions: A Dataset of Fine-Grained Emotions,” arXiv.org, May 01, 2020. https://arxiv.org/abs/2005.00547

Link to Dataset

https://huggingface.co/datasets/mrm8488/goemotions

Gantt Chart

Task	Wh
Project Team Composition (picking team members)	Eve
Project Proposal	
Introduction, background, literature review	Mic
Potential Datasets	Xin
Methotology	Mic
Results and Discussion	Mic
Video Creation	Yira
Github Pages and Submission	Jian
Midterm Reports	
Data Processing & ML method Design and Selection	Eve
Transformer-based tokenization and embedding	Xiw
Dataset and Result Visualization	Xin
Neural Network Implementation and Coding	Xiw
Neural Network Results Evaluation	Xin
Midterm Report	Eve
website update	Jian
Final Report	
Model selection	Eve
Improve NN model	Yira
Linear Regression Data Cleaning/Processing	Xiw
Transformer Implementation and Coding	Yira
Linear Regression Implementation and Coding	Xiw
Comparison of Algorithms	Jian
Video Creation	Xin
Final Report	Jian

Contribution Table

#	Name	Proposal Contributions
1	Xinyi Wang	Video Presentation
2	Jiangyue Zhu	Analyze models performance and Update GitHub website
3	Yiran Luo	Data Processing and Model Training(Transformer and NN)
4	Xiwen Zhang	Data Processing and Model Training(Logistic Regression)

Video Presentation

https://youtu.be/dxTQXYrnR94?si=aSVKed_H0sX27tOZ