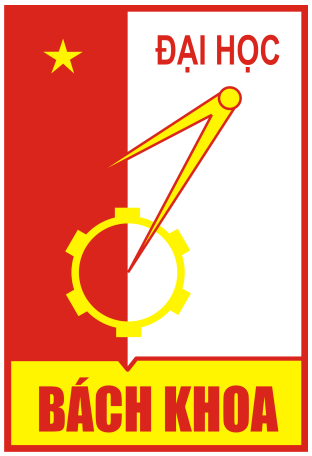
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



INTRODUCTION TO ARTIFICIAL INTELLIGENCE - IT3160E

CAPSTONE PROJECT: CORPORATE STOCK REVIEW

Instructor: Assoc. Prof. Pham Van Hai

Students: Nguyen Chi Long - 20210553
Ngo Xuan Bach - 20215181
Le Xuan Hieu - 20215201
Dinh Viet Quang - 20215235

Ha Noi, July - 2023

Contents

Abstract

Sentiment analysis plays a crucial role in understanding public opinion, provides valuable insights for investors, analysts, and other stakeholders, classifies it as positive or negative sentiment. It consists of the computer's ability to recognize, understand and predict even the most complex textual data by employing advanced natural language processing techniques. In this Corporate Stock Review project, we wish to apply Machine Learning and Deep Learning approaches to measure the accuracy of the model, highlight the advantages and disadvantages of each model and identify the best algorithm for Vietnamese sotck article classification. Furthermore, we performed extensive and intensive analysis on models and also tried a variety of approaches to reach better results. After trial and error, we finally got 82.04% accuracy on the split test set of dataset.

1 Introduction

Stock analysis, which has always been a crucial aspect of any organization, is a method for investors and traders to make buying and selling decisions. By studying and evaluating past and current data, investors and traders attempt to gain an edge in the markets by making informed decisions.

Sentiment analysis also known as opinion mining is a sub-field within Natural Language Processing (NLP) that builds machine learning algorithms to categorize the emotional polarity of a text, such as positive or negative opinions. Over the years, sentiment analysis has garnered a lot of attention and advancements in both academic and industry circles. The sentiment analysis task is often formulated as a classification problem, where a classifier is fed with a text and returns the corresponding sentiment label, e.g., positive, negative, or neutral. In other words, the problem of learning the sentimental polarities of opinions is reduced to a classification problem.

Given the problem and the dataset, in order to solve the above problem, we utilized many techniques of data preparation and AI models to find the optimal and effective solution. Speaking of tools, we chose Scrapy for collecting corporate stock reviews from various sources, while Scikit-learn, Pytorch, Tensorflow, Pandas, etc. were used to analyze the data, visualize graphs and build our machine models faster and easier

In the report, we presented the process we tackled the aforementioned problem step by step. Firstly we defined the issue more carefully. Secondly, we explored the dataset we found and collected to discover interesting patterns and results. Additionally, machine learning and deep learning models were built and the reasons why we chose them would be presented as well as methods and techniques to increase the efficiency of each model over time such as cross-validation, learning curve and so on. Finally, we considered testing the models through the Jupyter Notebook in the case of a single sentence and an article with multiple passages.

2 Problem Formulation

The problem of stock prediction through text data in newspaper articles involves leveraging natural language processing (NLP) techniques to extract relevant information from textual data and utilizing it to accurately forecast stock prices. The objective is to develop a system that can analyze news articles and identify key sentiments, events, and market indicators that may impact stock prices. By addressing this challenge, the system can provide investors, traders, and financial analysts with valuable insights for making informed decisions and optimizing their investment strategies based on the content of articles. This can be formulated as follows:

Given a dataset of articles that contains information related to corporate stocks, the goal is to build a predictive model that can utilize the textual data to accurately forecast the future prices of stocks. The model should employ machine learning or deep learning models to extract sentiment-related information, identify relevant events and news, and capture market indicators mentioned in the articles.

To achieve accurate predictions, the model should take into account the sentiment polarity of the articles, the tone of the language used, and the relevance and credibility of the news sources. It should also consider the potential impact of specific events or news on stock prices, such as mergers and acquisitions, earnings reports, regulatory changes, and market trends.

Evaluation of the stock prediction system should focus on assessing the accuracy of the forecasts using appropriate evaluation metrics, such as confusion matrix, f1-score or prediction accuracy within a certain threshold. The system's performance should be compared against baseline models and a benchmark dataset to determine its effectiveness.

In summary, the problem of stock prediction through text data in articles involves developing a predictive model that can analyze textual data, extract relevant information, and utilize it to accurately forecast stock prices. By considering sentiment analysis, event detection, and market indicators, the system aims to provide valuable insights for investors and analysts, enabling them to make informed decisions based on the information published in newspaper

articles. With the motivation to explore data processing methods and apply various models to this situation, we have made our best efforts to achieve the optimal results for this problem.

3 Data Preprocessing

3.1 Dataset

1. Data Collection: The desired information was collected from various sources using Scrapy, including major Vietnamese articles on business and finance such as CafeF, VnExpress, etc. under the .txt format. We then converted the obtained data to a JSON format for flexibility (e.g., nested fields,...) and performed labeling to ensure consistency across multiple sources and usability for further stages. The dataset required for our objective consists of sentences related to stocks and finance with a label corresponding to the nuanced content of the sentence.

2. Data Preprocessing:

   - Remove noise and irrelevant information: Remove stop words, special characters, punctuation marks, emojis, and any irrelevant information that does not contribute to sentiment analysis.
   - Split compound sentences: Split compound sentences while keeping the subject-verb pairs intact by checking stopwords. This step helps to break down complex sentences into simpler units for better analysis.
   - Tokenization: connect Vietnamese words[12]. Then, convert the text into individual words or tokens, which is replacing sensitive data with unique identification symbols that retain all the essential information about the data without compromising its security. This step is important for further analysis and feature extraction.

3. Data Labeling: We used a tool called Doccano to perform data labeling by annotating one or more predefined labels or tags (e.g., "positive," "negative," or "neutral") to each set of customer reviews. This step is crucial for training a machine learning model for sentiment analysis. Specifically, in the field of stocks and securities, positive feedback often relates to the development of a business or a stock, or it can also reflect the success that the business has achieved as mentioned in the article. For negative feedback, economic downturns and stock value declines are mentioned, which in turn leads to consequences for the way the business operates. These feedbacks can also be the negative behaviors of an individual or an organization that adversely affects the economy and society. Finally, neutral comments often refers to basic information and statistics that cannot be deduced from the development or downgrade of the business.

   - An example of a review which is classified as “Positive”: Ngay thời điểm hiện tại, cũng có những nhóm ngành trên thị trường chưa cần đến tháng 4 thì đã đón nhận nhiều thông tin tốt như bất động sản, chủ yếu về mặt chính sách bao gồm các Nghị định sửa đổi, các Quyết định của Chính phủ và các thông tư hỗ trợ điều hành của Ngân hàng Nhà nước.
   - An example of a review which is classified as “Negative”: Theo Uỷ ban, lý do của Vietnam Airlines để xin chậm công bố báo cáo tài chính kiểm toán năm 2022 không thuộc trường hợp được tạm hoãn công bố thông tin như thiên tai, hoả hoạn, dịch bệnh, chiến tranh hoặc các lý do bất khả kháng khác.
   - An example of a review which is classified as “Neutral”: Trong năm 2023, Công ty đặt kế hoạch khai trương thêm 2 trung tâm thương mại và dự kiến bổ sung hơn 800 nghìn m2 diện tích sàn bán lẻ vào danh mục dự án trong ba năm tiếp theo, theo đó VMM The Empire và Đông Anh Cổ Loa tại Hà Nội, VCP Vinh tại tỉnh Nghệ An và VCP Bắc Giang tại tỉnh Bắc Giang là một số trung tâm thương mại sắp khánh thành.

3.2 Data Exploration and Visualization

This section’s work will focus on the dataset for training models. Once the data has been prepared, we can perform data exploration and visualization to gain insights into the sentiment of the stock reviews. In this project, we only worked with the training set and made predictions on the test set. With a given training dataset, competitors are encouraged to build a predictive model and apply it to the test set to predict the sentiment of a sentence or a passage.

By active learning process, we ended up with a dataset split with 1516 training samples and 1002 testing

samples. Additionally, we augmented our training dataset with visualized statistics.

The dataset consists of sentences labeled as positive, negative, or neutral, and their lengths follow a chi-squared distribution for all three labels. The chi-squared distribution is valuable because it allows for quick significance testing, particularly in machine learning algorithms. Since it is a non-parametric test, assumptions about the data are not necessary. The graph reveals that the majority of sentences fall within the 15 to 25-word range, although the proportions for each length are relatively low, approximately 3% to 5%. Moreover, the distribution of sentence lengths appears to be similar between the training and test dataset.

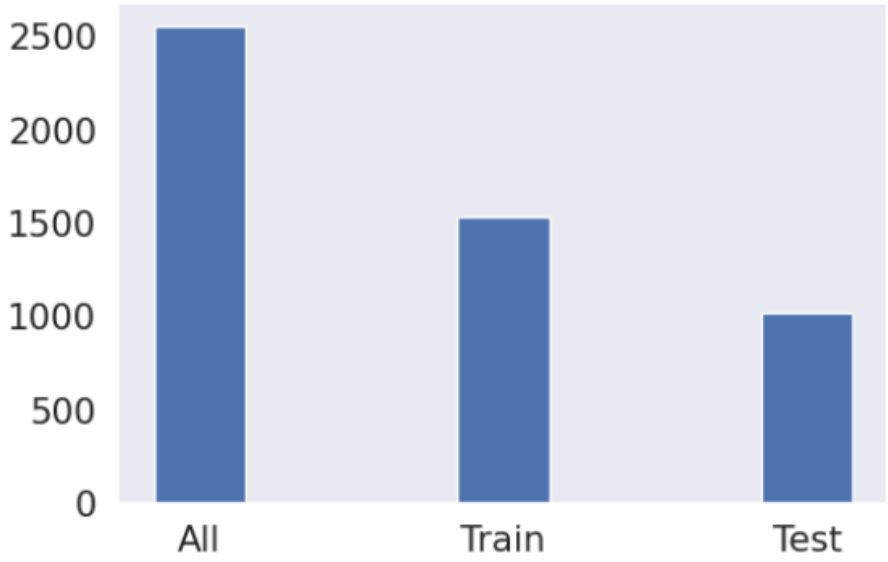


Figure 1: Number of samples

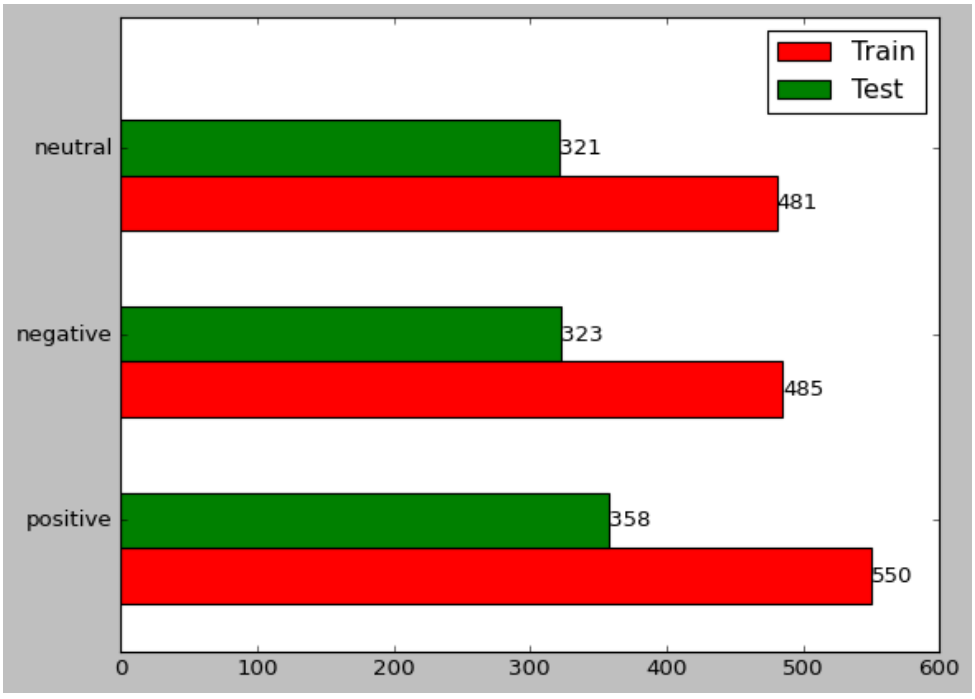
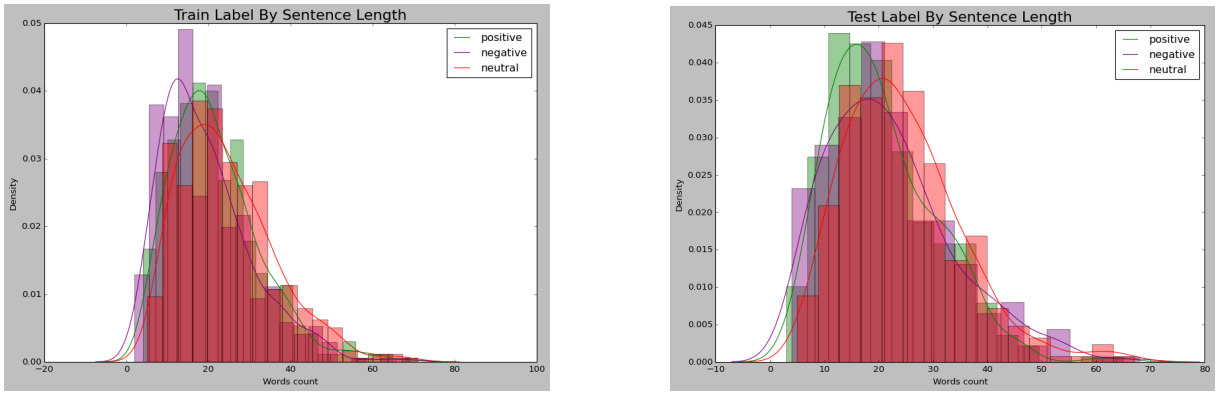


Figure 2: Class distribution



(a) Training dataset **(b)** Test dataset

Figure 3: Sentence Length distribution

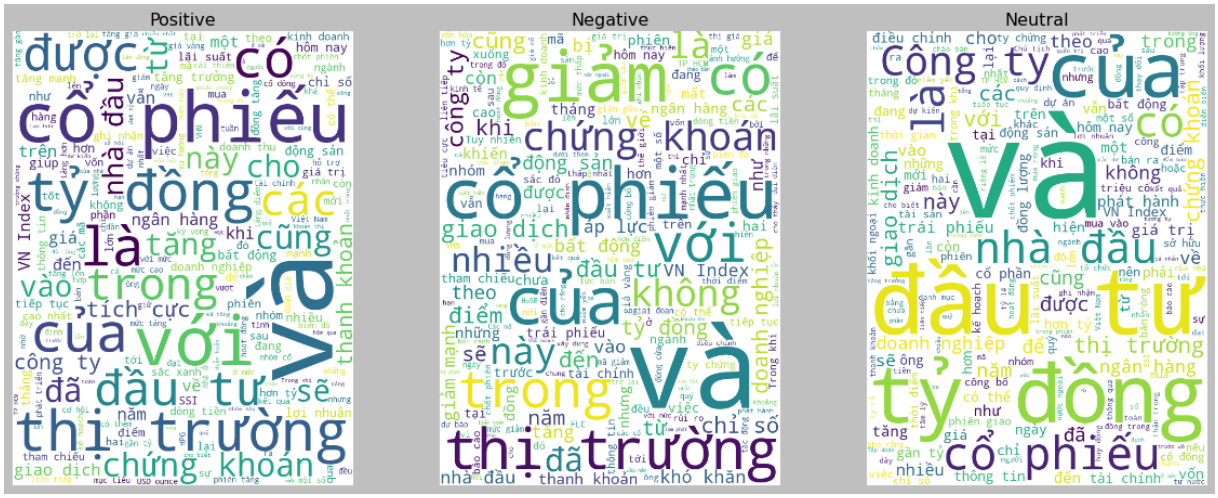


Figure 4: Wordcloud

We also want to explore the insights of each dataset label by using wordcloud. Therefore, Word frequency analysis provides insights into the most commonly occurring words in different sentiment labels (positive, negative, and neutral) within the dataset. By visualizing these word frequencies using a wordcloud, we can easily identify the key terms associated with each sentiment label. The size and prominence of each word in the wordcloud are proportional to its frequency within the respective sentiment label. Consequently, larger and bolder words indicate more frequent occurrences, while smaller and fainter words signify less frequent appearances.

First, we can see the common point in all three data sets is the appearance of common terms in the field of stocks such as “cổ phiếu”, “thị trường”, ”tỷ đồng”. We can consider these as stopwords, which are commonly used words in a language that do not carry significant meaning. We should identify as many stopwords as possible since this is undoubtedly one of the most important steps to increase the accuracy rate in AI model training.

In the positive sentiment label, the wordcloud displays prominent words that reflect optimism, satisfaction, and positive experiences. Words such as “đồng tăng”, “đầu tư”, “tích cực” and “phiên tăng” are likely to appear prominently, indicating positive sentiment in the dataset. These words contribute to an overall positive tone and highlight favorable aspects related to the subject matter.

In the negative sentiment label, the wordcloud reveals words that convey dissatisfaction, criticism, and negative experiences. Terms like “giảm”, “khó khăn”, “tiêu cực” and “sắc đỏ” are likely to be dominant, signifying negative sentiment in the dataset. These words reflect negative aspects or unfavorable opinions associated with the subject matter.

For the neutral sentiment label, the wordcloud exhibits words that are neither strongly positive nor strongly negative. They represent a lack of strong emotional inclination towards the subject matter, indicating a neutral sentiment in the dataset.

Analyzing the word frequencies in positive, negative, and neutral sentiment labels through wordcloud visu-

alization enables a quick and intuitive understanding of the prevailing sentiments associated with the dataset. It facilitates identifying common themes, sentiments, and trends, thereby aiding in sentiment analysis and providing valuable insights for further analysis and decision-making.

4 Methods

In this section, we will clarify the reason why we selected each following models and present the underlying theories of these models. Understanding the underlying theories can help us address this challenging problem more effectively. We start from machine learning model, namely Random Forest, combined with TF - IDF techniques. Next, we will go through many more advanced models like Deep Bi-directional Neural Network and Bidirectional Encoder Representations from Transformers (BERT) in order to find out which model achieves the highest accuracy

4.1 Term Frequency-Inverse Document Frequency (TF - IDF) - Random Forest

4.1.1 TF - IDF

TF - IDF is a widely used technique in natural language processing (NLP) for assessing the importance of words in a document collection. It calculates a numerical value for each word that represents how relevant it is to a particular document within a larger collection of documents. TF - IDF is particularly useful in information ret rieval, text mining, and document classification tasks.

In a large text corpus, some words will be very present (e.g. “the”, “a”, “is” in English) hence carrying very little meaningful information about the actual contents of the document. If we were to feed the direct count data directly to a classifier like [2], those very frequent terms would shadow the frequencies of rarer yet more interesting terms.

In order to re-weight the count features into floating point values that are suitable for usage by a classifier, it is very common to use the **TF - IDF** transform. *tf* means term-frequency while *tf idf* means term-frequency times inverse document-frequency:

$$\mathbf{tfidf(t, d)} = \mathbf{tf(t, d)} \times \mathbf{idf(t)} \tag{1}$$

tf(t, d) is the number of times the word *t* occurs in a document *d*. *idf(t)* is employed to determine a word’s importance. When calculating the frequency of occurrence *tf*, the words are considered equally important. Although they are often used, some words are not necessary to convey the document’s content. Therefore, we must use *idf* to lessen the importance of such words.

$$\mathbf{idf(t)} = \log \frac{\mathbf{n}}{\mathbf{df(t)}} + \mathbf{1} \tag{2}$$

where *n* is the total number of documents in the document set, and *df(t)* is the number of documents in the document set that contains term *t*.

$$v_{\text{norm}} = \frac{v}{\|v\|^2} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}} \tag{3}$$

The resulting ***TF - IDF*** vectors are then normalized by the Euclidean norm above. The TF - IDF score for a specific word in a document is obtained by multiplying its Term Frequency (TF) by its Inverse Document Frequency (IDF). The resulting score assigns higher importance to words that appear frequently within a specific document but are rare in the overall collection.

4.1.2 Random Forest

Random Forest is an ensemble method used for classification and a well-known popular machine learning algorithm used for both classification and regression tasks. It is also the first technique we want to introduce about ensemble learning, which is bagging as it could reduce the variance of the previous model and is one of the most popular applications. In Random Forest, we grow multiple trees as opposed to a single tree in Decision Tree model. One of the major concerns with decision trees is overfitting, which results in an extremely poor predictive model. Adding more trees to the random forest eliminates overfitting and results in a much better predictive model. Each tree offers a classification to categorize a new object based on its characteristics, and we say the tree “votes” for that class. The categorization with the highest votes is selected by the forest (over all the trees in the forest). Random

forest algorithms have three main hyperparameters, which need to be set before training. These include node size, the number of trees, and the number of features sampled.

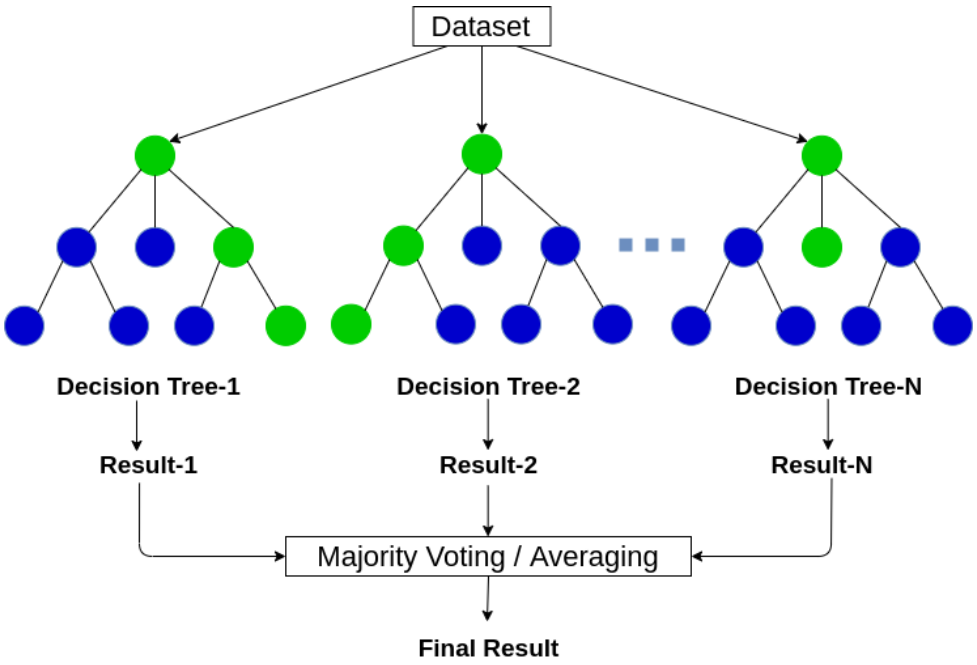


Figure 5: Random Forest

We may employ Random Forest in this problem after TF - IDF model training. Following the Tf and idf-based processing approach, the input text segments are transformed into vectors of the same dimension, where each word represents a dataset feature. To evaluate the quality of a split, we will use the Gini function.

$$\mathbf{G} = \sum_{k=1}^{K} p_{mk} \left(1 - p_{mk}\right) \tag{4}$$

where p_{mk} represents the proportion of observations in the m_{th} region from the k_{th} class. In essence, the Gini function is a measure of variance. The higher the variance, the more misclassification there is. Therefore lower values of the Gini criterion yield best classification.

	precision	recall	f1-score	support
negative	0.77	0.69	0.73	323
neutral	0.59	0.69	0.64	321
positive	0.73	0.68	0.70	358
accuracy			0.69	1002
macro avg	0.70	0.69	0.69	1002
weighted avg	0.70	0.69	0.69	1002

Train Accuracy: 0.996042216358839
Test Accuracy: 0.687624750499002

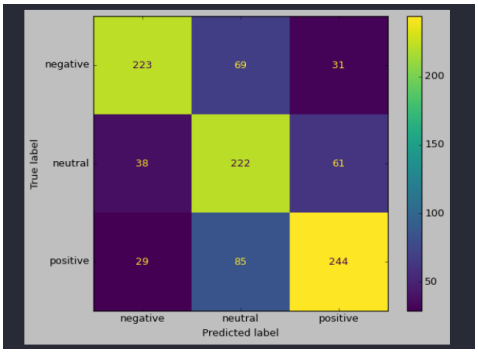


Figure 6: The classification report for Random Forest model with TF - IDF techniques

4.2 Deep Bi-directional Network

4.2.1 Long Short-Term Memory (LSTM)

The challenge to address long-term information preservation and short-term input skipping in latent variable models has existed for a long time. One of the earliest approaches to address this was long short-term memory (LSTM). It shares many of the properties of the GRU. Interestingly, LSTMs have a slightly more complex design than GRUs but predate GRUs by almost two decades.

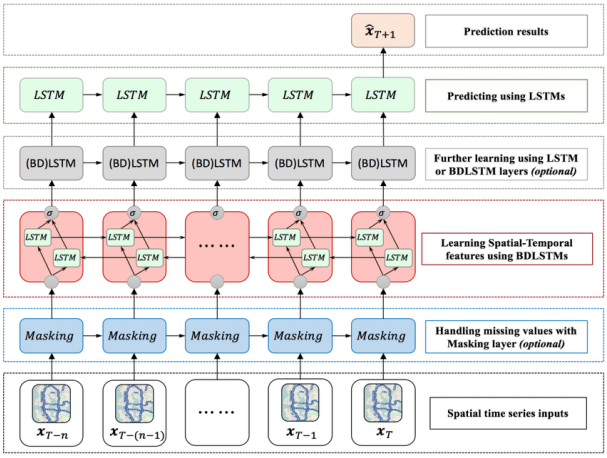


Figure 7: Deep Bidirectional Long Short Term Memory

Arguably LSTM's design is inspired by the logic gates of a computer. LSTM introduces a memory cell (or cell for short) that has the same shape as the hidden state (some literature considers the memory cell as a special type of the hidden state), engineered to record additional information. To control the memory cell we need a number of gates. One gate, called the output gate, is required to read out the entries from the cell. A second gate is needed to decide when to read data into the cell. We refer to this as the input gate. Last, we need a mechanism to reset the content of the cell, governed by a forget gate. The motivation for such a design is the same as that of GRUs, namely to be able to decide when to remember and when to ignore inputs in the hidden state via a dedicated mechanism.

The data feeding into the LSTM gates are the input at the current time step and the hidden state of the previous time step, as illustrated in this figure.

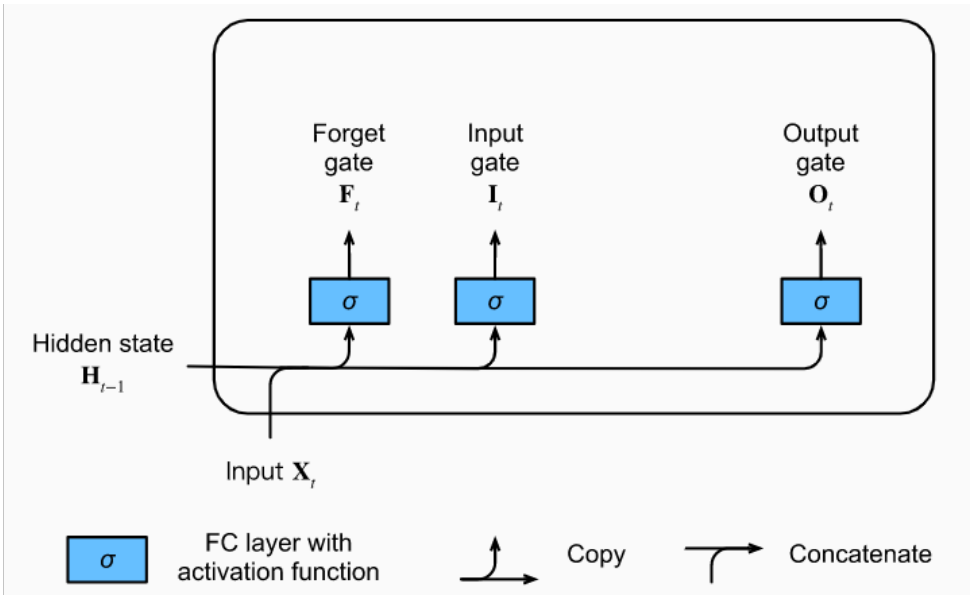


Figure 8: Computing the input gate, the forget gate, and the output gate in an LSTM model

Mathematically, suppose that there are h hidden units, the batch size is n, and the number of inputs is d. Thus, the input is $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ and the hidden state of the previous time step is $\mathbf{H}_{t-1} \in \mathbb{R}^{n \times h}$. Correspondingly, the gates at time step t are defined as follows: the input gate is $\mathbf{I}_t \in \mathbb{R}^{n \times h}$, the forget gate is $\mathbf{F}_t \in \mathbb{R}^{n \times h}$, and the output gate is $\mathbf{O}_t \in \mathbb{R}^{n \times h}$. They are calculated as follows:

$$\begin{aligned}\mathbf{I}_t &= \sigma\left(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i\right)\\ \mathbf{F}_t &= \sigma\left(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f\right)\\ \mathbf{O}_t &= \sigma\left(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o\right)\end{aligned}$$

where $\mathbf{W}_{xi}, \mathbf{W}_{xf}, \mathbf{W}_{xo} \in \mathbb{R}^{d \times h}$ and $\mathbf{W}_{hi}, \mathbf{W}_{hf}, \mathbf{W}_{ho} \in \mathbb{R}^{h \times h}$ are weight parameters and $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^{1 \times h}$ are bias parameters.
Candidate Memory Cell: Next, we design the memory cell. Since we have not specified the action of various gates yet, we first introduce the candidate memory cell $\tilde{\mathbf{C}}_t \in \mathbb{R}^{n \times h}$. Its computation is similar to that of three gates described above, but using a tanh function with a value range for $(-1, 1)$ as the activation function. This leads to the following equation at time step t :

$$\tilde{\mathbf{C}}_t = \tanh\left(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c\right)$$

where $\mathbf{W}_{xc} \in \mathbb{R}^{d \times h}$ and $\mathbf{W}_{hc} \in \mathbb{R}^{h \times h}$ are weight parameters and $\mathbf{b}_c \in \mathbb{R}^{1 \times h}$ is a bias parameter. A quick illustration of the candidate memory cell is shown in the below figure

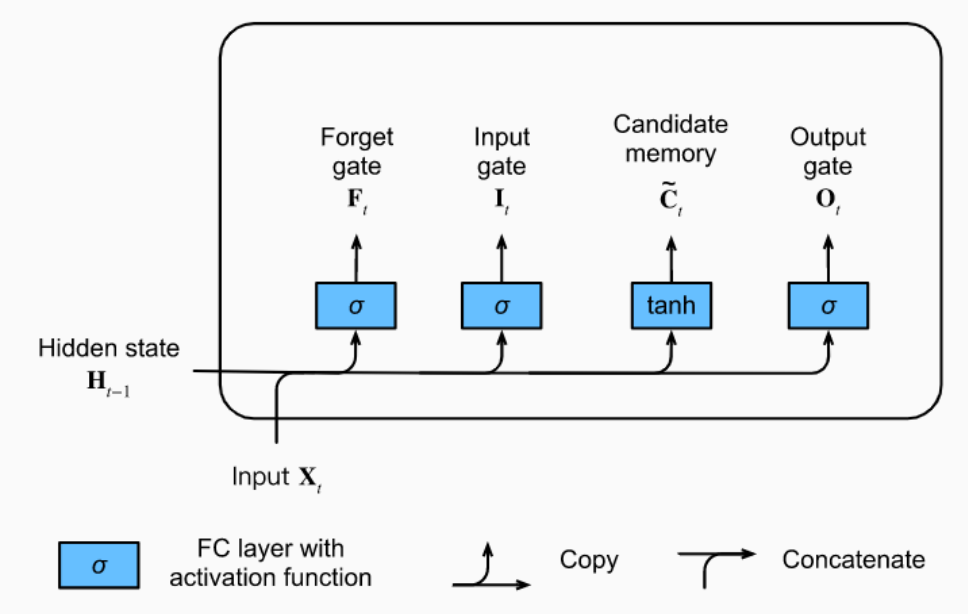


Figure 9: Computing the candidate memory cell in an LSTM model

Memory cell: In GRUs, we have a mechanism to govern input and forgetting (or skipping). Similarly, in LSTMs we have two dedicated gates for such purposes: the input gate $\mathbf{I}_t$ governs how much we take new data into account via $\tilde{\mathbf{C}}_t$ and the forget gate $\mathbf{F}_t$ addresses how much of the old memory cell content $\mathbf{C}_{t-1} \in \mathbb{R}^{n \times h}$ we retain. Using the same pointwise multiplication trick as before, we arrive at the following updated equation:

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t$$

If the forget gate is always approximately 1 and the input gate is always approximately 0, the past memory cells $\mathbf{C}_{t-1}$ will be saved over time and passed to the current time step. This design is introduced to alleviate the vanishing gradient problem and to better capture long-range dependencies within sequences. We thus arrive at the flow diagram:

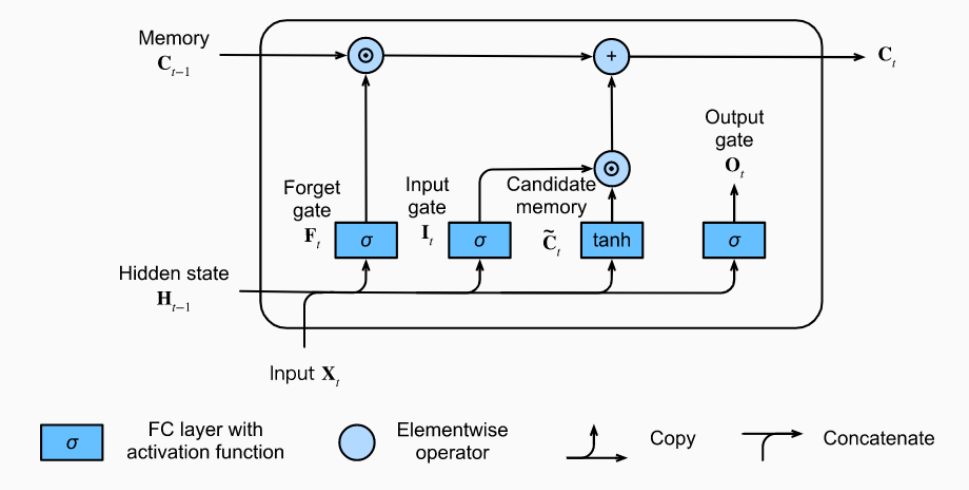


Figure 10: Computing the memory cell in an LSTM model

Hidden State: Last, we need to define how to compute the hidden state $\mathbf{H}_t \in \mathbb{R}^{n \times h}$. This is where the output gate comes into play. In LSTM it is simply a gated version of the tanh of the memory cell. This ensures that the values of $\mathbf{H}_t$ are always in the interval $(-1, 1)$.

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh\left(\mathbf{C}_t\right).$$

Whenever the output gate approximates 1 we effectively pass all memory information through to the predictor, whereas for the output gate close to 0 we retain all the information only within the memory cell and perform no further processing.

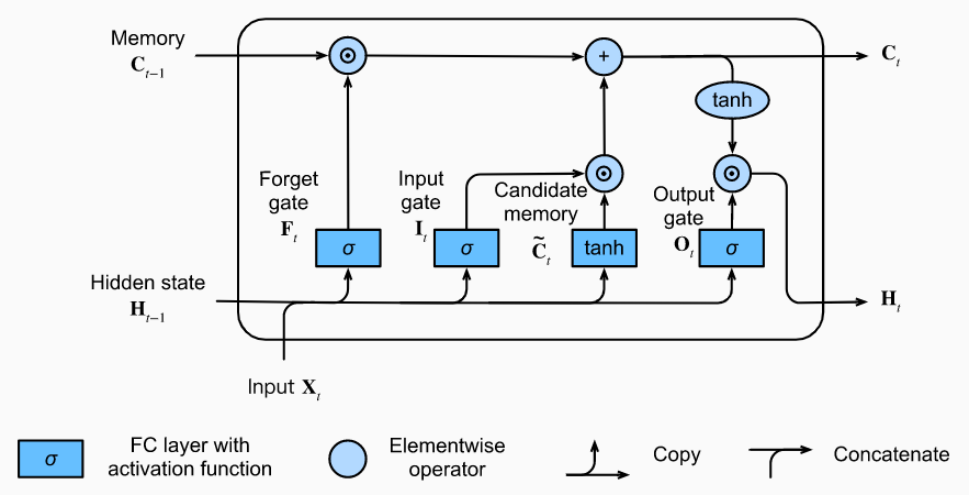


Figure 11: Computing the hidden state in an LSTM model

4.2.2 Gated Recurrent Units (GRU)

Bidirectional RNNs were introduced by [8]. For a detailed discussion of the various architectures see also the paper [5]. Let's look at the specifics of such a network.

For any time step t, given a minibatch input $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ (number of examples: n, number of inputs in each example: d) and let the hidden layer activation function be ϕ. In the bidirectional architecture, we assume that the forward and backward hidden states for this time step are $\overrightarrow{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$ and $\overleftarrow{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$, respectively, where h is

the number of hidden units. The forward and backward hidden state updates are as follows:

$$\overrightarrow{\mathbf{H}}_t = \phi\left(\mathbf{X}_t \mathbf{W}_{xh}^{(f)} + \overrightarrow{\mathbf{H}}_{t-1} \mathbf{W}_{hh}^{(f)} + \mathbf{b}_h^{(f)}\right)$$
$$\overleftarrow{\mathbf{H}}_t = \phi\left(\mathbf{X}_t \mathbf{W}_{xh}^{(b)} + \overleftarrow{\mathbf{H}}_{t+1} \mathbf{W}_{hh}^{(b)} + \mathbf{b}_h^{(b)}\right)$$

where the weights $\mathbf{W}_{xh}^{(f)} \in \mathbb{R}^{d \times h}, \mathbf{W}_{hh}^{(f)} \in \mathbb{R}^{h \times h}, \mathbf{W}_{xh}^{(b)} \in \mathbb{R}^{d \times h}$, and $\mathbf{W}_{hh}^{(b)} \in \mathbb{R}^{h \times h}$, and biases $\mathbf{b}_h^{(f)} \in \mathbb{R}^{1 \times h}$ and $\mathbf{b}_h^{(b)} \in \mathbb{R}^{1 \times h}$ are all the model parameters. Next, we concatenate the forward and backward hidden states $\overrightarrow{\mathbf{H}}_t$ and $\overleftarrow{\mathbf{H}}_t$ to obtain the hidden state $\mathbf{H}_t \in \mathbb{R}^{n \times 2h}$ to be fed into the output layer. In deep bidirectional RNNs with multiple hidden layers, such information is passed on as input to the next bidirectional layer. Last, the output layer computes the output $\mathbf{O}_t \in \mathbb{R}^{n \times q}$ (number of outputs: q):

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q$$

Here, the weight matrix $\mathbf{W}_{hq} \in \mathbb{R}^{2h \times q}$ and the bias $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$ are the model parameters of the output layer. In fact, the two directions can have different numbers of hidden units.

4.2.3 Bi-directional

Researchers have developed more sophisticated types of Recurrent Neural Networks (RNN) to deal with the shortcomings of the standard RNN model: Bidirectional RNN, Deep Bidirectional RNN and Long Short Term Memory (LSTM) network. Bidirectional RNN is based on the idea that the output at each time step may not only depend on the previous elements in the sequence but also depend on the subsequent elements in the sequence. For instance, to predict a missing word in a sequence, we may need to look at both the left and the right context. A bidirectional RNN24 consists of two RNNs, which are stacked on top of each other. A RNN that processes the input in its original order and the other one processes the reversed input sequence. The output is then computed based on the hidden state of both RNNs. Deep bidirectional RNN is similar to bidirectional RNN. The only difference is that it has multiple layers per time step, which provides higher learning capacity but requires a larger amount of training data.

```
12/12 [==============================] - 1s 73ms/step - loss: 0.0019 - acc: 0.9980
8/8 [==============================] - 1s 102ms/step - loss: 2.0948 - acc: 0.6228
32/32 [==============================] - 5s 16ms/step
              precision    recall  f1-score   support

    positive       0.68      0.64      0.66       358
    negative       0.70      0.61      0.65       323
     neutral       0.52      0.62      0.56       321

    accuracy                           0.62      1002
   macro avg       0.63      0.62      0.62      1002
weighted avg       0.63      0.62      0.63      1002
```

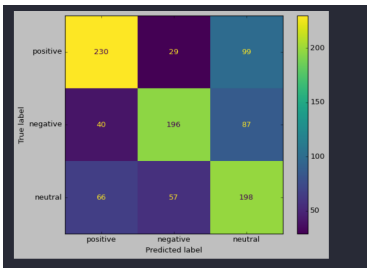


Figure 12: The classification report for Deep Biderectional Long Short-term Memory

```
12/12 [==============================] - 1s 51ms/step - loss: 0.0019 - acc: 0.9980
8/8 [==============================] - 0s 60ms/step - loss: 1.5842 - acc: 0.6367
32/32 [==============================] - 5s 13ms/step
              precision    recall  f1-score   support

    positive       0.70      0.64      0.67       358
    negative       0.72      0.66      0.69       323
     neutral       0.52      0.60      0.56       321

    accuracy                           0.64      1002
   macro avg       0.64      0.64      0.64      1002
weighted avg       0.65      0.64      0.64      1002
```

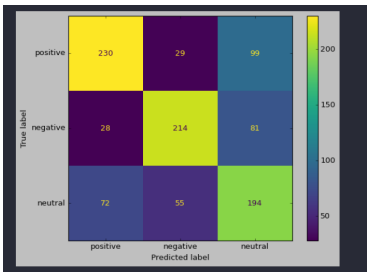


Figure 13: The classification report for Deep Biderectional Gated Recurrent Units

4.3 PhoBERT (Pre-trained language model for Vietnamese)

Published in [7], this is a variant of BERT (Bidirectional Encoder Representations from Transformers) architecture that is specifically designed and trained for the Vietnamese language. PhoBERT is known as a deep neural network model that is pre-trained on a large corpus of Vietnamese text data, which allows it to learn the underlying patterns and structure of the language.

The architecture of PhoBERT is the same as that one of BERT, and the pre-training process is based on RoBERTa. This model was pre-trained by 2 datasets: the Vietnamese Wikipedia corpus ($\sim$ 1 GB) and a second corpus ($\sim$ 19 GB) which is generated by removing similar articles and duplication from a 50 GB Vietnamese news corpus.

BERT is first introduced in [3]. BERT has 2 versions which have the same kind of architecture: $\text{BERT}_{\text{base}}$ and $\text{BERT}_{\text{large}}$. The $\text{BERT}_{\text{base}}$ model uses L = 12 layers of transformers block with a hidden size of H = 768 and number of self-attention heads as A = 12 and has around 110M trainable parameters. On the other hand, $\text{BERT}_{\text{large}}$ uses L = 24 layers of transformers block with a hidden size of H = 1024 and number of self-attention heads as A = 16 and has around 340M trainable parameters. (Note: S = 512: the maximum size of (maximum number of words in) a sentence). In the next part, we will explain the architecture of BERT and how RoBERTa pre-trained it.

Before discussing BERT layers, it is important to mention that a bias parameter will be added to each value in the output vector of every layer.

Input embedding: To prepare the input for BERT, the sentences are tokenized by VnCoreNLP, a library written in Python by PhoBERT's author. Then we add a special [CLS] token at the beginning of sentence A and a [SEP] token to separate the two sentences in a sentence pair (A,B). You can refer to Figure 14 to see an example of this. The input embedding vector has a size of H and is calculated as the sum of three different embedding vectors. The first is the token embedding vector, which is based on WordPiece embeddings and uses a vocabulary of over 30,000 tokens. The second is the segment embedding vector, which assigns the same embedding vector to any two words in the same segment. Finally, the position embedding vector is based on the position of each word in the concatenation of the two sentences.

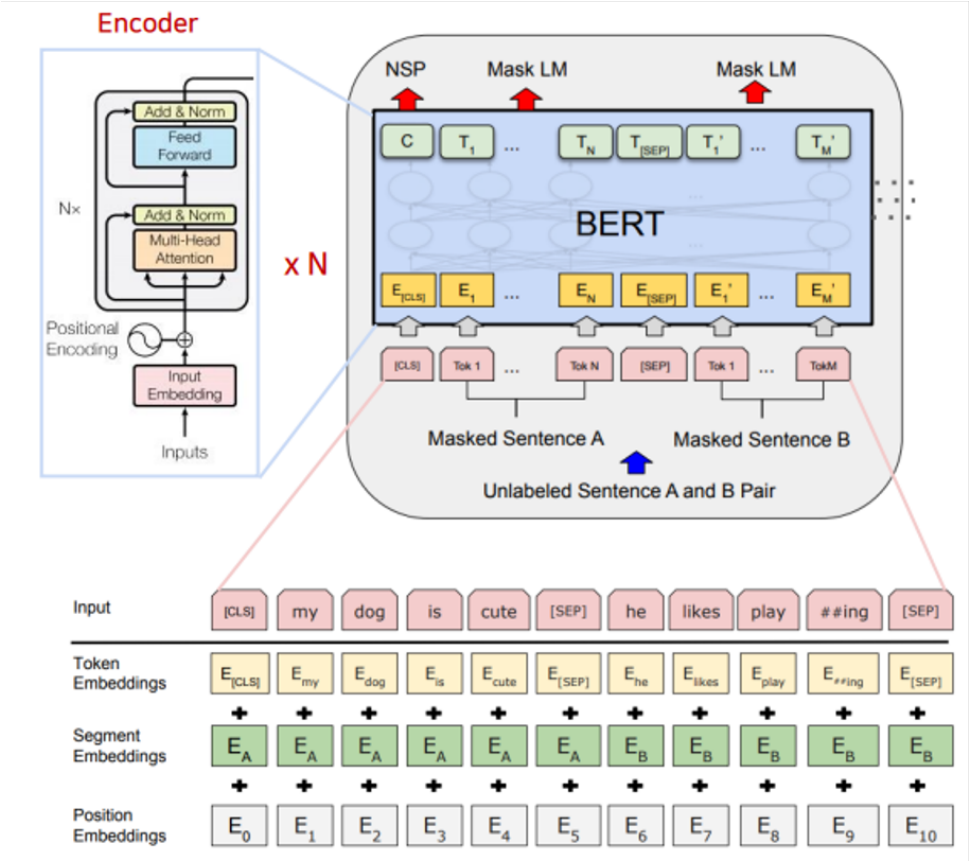


Figure 14: BERT Architecture

Encoder layer: BERT is essentially a Transformer Encoder stack of size L that has been trained. During the embedding process, each word embedding vector is padded, resulting in an input matrix X with a size of S*H for each sentence. In the encoder layer, we will talk about the sublayers that make up the stack.

+ Multi-head attention: This layer is first mentioned in [11], and it is based on the self-attention layer, so we will infer to self-attention layer.

Self-attention layer: The self-attention layer is based on the concept that certain words in a sentence are more important than others for a particular task or objective. This layer is designed to identify and prioritize these important words by using the query, key, and value matrices to perform self-attention on the input matrix.

Given an input matrix X, which consists of three parameter matrices: query matrix: W^Q, key matrix: W^K, value matrix: W^V. All three have the same size of H*d_k, where $d_k = \frac{H}{A}$. The layer will generate an output matrix Z with a size of S*d_k, and here's the function:

$$Z = \text{softmax}\left(\frac{Q * K^T}{\sqrt{d_k}}\right) * V$$

where $Q = X * W^Q$, $K = X * W^K$, $V = X * W^V$.

Multi-head attention layer: Based on the idea that a single word's meaning maybe different in different contexts, multi-head attention was born.

Because of using multi-head, we will use 3*A parameter matrices: $(W_0^Q, W_0^K, W_0^V), \ldots, (W_{A-1}^Q, W_{A-1}^K, W_{A-1}^V)$. After applying the same process as in the self-attention layer, we will obtain A output matrices, denoted as $(Z_0, \ldots, Z_{A-1})$. Then, because we need to get an output vector corresponding to each word, we concatenate these matrices to form a matrix $Z_0 Z_1 \ldots Z_{A-1}$, and multiply it with a parameter matrix W^0 with a size of H*H to get the output matrix Z.

+ Feed forward: This is truly a double fully-connected layer, such that each vector of size H is fully-connected to a vector of size 4*H, which is then fully-connected to a vector of size H. Both layers use the ReLU activation function.

+ Add & Norm: We can see that, after each of these 2 layers, there's an Add & Norm layer. For each vector x, it returns LayerNorm(x+Sublayer(x)), where Sublayer is either Multi-head Attention or Feed Forward. The LayerNorm function is defined as:

$$\text{LayerNorm}(x_j) = \gamma_j * \frac{x - \mathbb{E}[x]}{\sqrt{\text{var}(x) + \epsilon}} + \beta_j$$

For classifier task, there's one more classifier that contains a feed-forward neural network and a softmax activation function.

- RoBERTa pre-training process:

+ BERT pre-training process: During pre-training, BERT uses two objectives: masked language modeling and next sentence prediction.

Masked Language Model (MLM): A random sample of the tokens in the input sequence is selected and replaced with the special token [MASK]. The MLM objective is a cross-entropy loss in predicting the masked tokens. BERT uniformly selects 15% of the input tokens for possible replacement. Of the selected tokens, 80% are replaced with [MASK], 10% are left unchanged, and 10% are replaced by a randomly selected vocabulary token.

Next Sentence Prediction (NSP): NSP is a binary classification loss for predicting whether two segments follow each other in the original text. Positive examples are created by taking consecutive sentences from the text corpus. Negative examples are created by pairing segments from different documents. Positive and negative examples are sampled with equal probability.

BERT is optimized with Adam using the following parameters: β_1 = 0.9, β_2 = 0.999, ϵ = 1e-6 and L_2 weight decay of 0.01. The learning rate is warmed up over the first 10,000 steps to a peak value of 1e-4, and then linearly decayed. BERT trains with a dropout of 0.1 on all layers and attention weights, and a GELU activation function $(g(x) = x\Phi_x)$. Models are pretrained for S = 1,000,000 updates, with mini batches containing B = 256 sequences of maximum length T = 512 tokens.

+ Pre-training process introduced in RoBERTa: RoBERTa was created by aggregating several improvements, including dynamic masking, FULL-SENTENCES without NSP task loss, large mini-batches and a larger byte-level BPE. The pre-training process for RoBERTa involved using 160GB of data from various sources, including BOOKCORPUS [13] (16GB), CC-NEWS (76GB), OPENWEBTEST (38GB), STORIES[10] (31GB).

Dynamic masking: In BERT pre-training process, mask token's index is chosen from the start and does not change during the procedure. However, in RoBERTa, mask tokens are randomly chosen in every batch.

Algorithm 1 Adam algorithm

Require: $\gamma \geq 0$(learning rate), $\beta_1, \beta_2, \theta_0, f(\theta)$(objective), λ(weight decay)
  $m_0 \leftarrow 0$
  $v_0 \leftarrow 0$
  $\widehat{v_0}^{\max} \leftarrow 0$
  for $t \leftarrow 1$ to T **do**
    if maximize **then**
      $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
    else
      $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    end if
    if $\lambda \neq 0$ **then**
      $g_t \leftarrow g_t + \lambda\theta_{t-1}$
    end if
    $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)g_t^2$
    $\widehat{m_t} \leftarrow m_t/(1-\beta_1^t)$
    $\widehat{v_t} \leftarrow v_t/(1-\beta_2^t)$
    $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t}/(\sqrt{\widehat{v_t}} + \epsilon)$
  end for
  return θ_t

Large mini-batches: The size of each batch is multiplied, and the number of batches is divided, so that the amount of training data remains the same.

Larger byte-level BPE: First introduced in [4], BPE is used to reduce the size of tokens.

Algorithm 2 Byte Pair Encoding (BPE)

procedure BPE(corpus)
  vocab ← initialize_vocab(corpus)
  while No more pair is in more than 1 word **do**
    pair_counts ← count_pairs(corpus, vocab)
    most_frequent ← find_most_frequent(pair_counts)
    vocab ← merge_pair(vocab, most_frequent)
    corpus ← replace_pair(corpus, most_frequent)
  end while
  return vocab
end procedure

In [6], the authors proposed modifications to the BERT pre-training procedure that improve end-task performance. RoBERTa was created by aggregating these improvements, evaluating their combined impact and using more data and more training steps.

	precision	recall	f1-score	support
positive	0.8371	0.8184	0.8277	358
negative	0.8734	0.8545	0.8638	323
neutral	0.7292	0.7632	0.7458	321
accuracy			0.8124	1002
macro avg	0.8132	0.8121	0.8124	1002
weighted avg	0.8142	0.8124	0.8131	1002

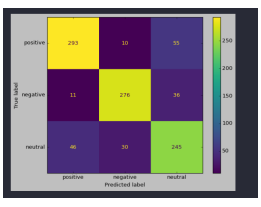


Figure 15: The classification report for phoBERT

4.4 XLM - RoBERTa

XLM - RoBERTa (Cross-lingual Language Model - RoBERTa), which is first introduced in [1], is a state-of-the-art pre-trained language model developed by Facebook AI Research. It is a variant of the popular RoBERTa

model that has been extended to handle cross-lingual tasks, making it particularly useful for multilingual natural language processing (NLP) applications.

XLM is a Transformer based architecture, which are scaling to 100 languagues, and is pre-trained using one of three language modelling objectives:

- Causal Language Modeling - models the probability of a word given the previous words in a sentence. This is the regular Language Modeling objective where we maximize the probability of a token x_t to appear at the t^{th} position in a given sequence given all the tokens $\boldsymbol{x}_{<t}$ (all the tokens preceding the t^{th} token) in that sequence:

$$\min_{\theta} logp_{\theta}(\boldsymbol{x}) = \sum_{t=1}^{T} logp_{\theta}(x_t|\boldsymbol{x}_{<t})$$

- Masked Language Modeling - the masked language modeling objective of BERT.
- Translation Language Modeling - a (new) translation language modeling objective for improving cross-lingual pre-training. We take a sequence of parallel sentences from the translation data and randomly mask tokens from the source as well as from the target sentence, like in this example:

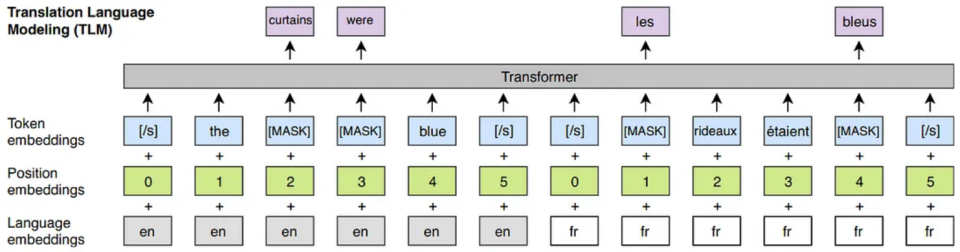


Figure 16: TLM

For this model, XLM model are pre-trained over RoBERTa architecture above, and then fine-tuned by XNLI (Cross-lingual Natural Language Inference) (objectives: translate-test: Dev and test sets are translated to English, translate-train: The English training set is machine-translated to each language, translate-train-all: The multilingual model is finetuned on a concatenation of all the training sets from translate-train), NER (Named Entity Recognition) (Fine-tuning ways: Trained on the English set to evaluate cross-lingual transfer, On each set to evaluate per-language performance, On all sets to evaluate multi-lingual learning) and Cross-lingual Question Answering. XLM-RoBERTa is a Language Model, so it is evaluated on the standard GLUE Benchmark.

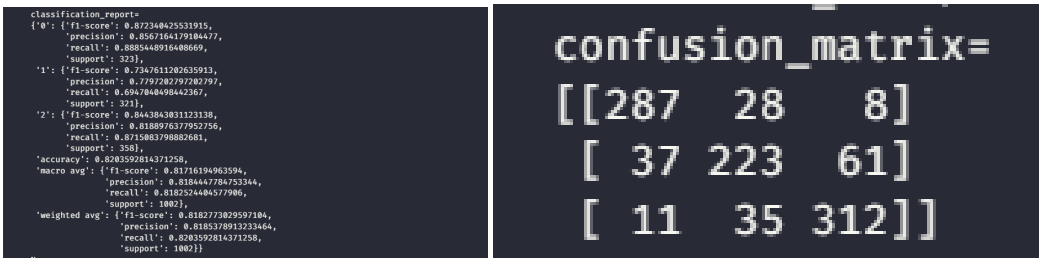


Figure 17: The classification report for XLM - RoBERTa

5 Experiments

This section will describe how we applied theories with previous sections into practice. Starting from the training strategy in subsection 5.1, where we describe how to prepare the data, what hyperparameters we chose for each model and which techniques we tried to improve our results, and in subsection 5.2, we will present our results in both quantitative and qualitative matters.

5.1 Training strategy

Our classification workflow is broken down into following steps:

1. Data Crawling
2. Data Annotation
3. Preprocessing
4. Feature extraction or Tokenization
5. Modelling and Training
6. Evaluation

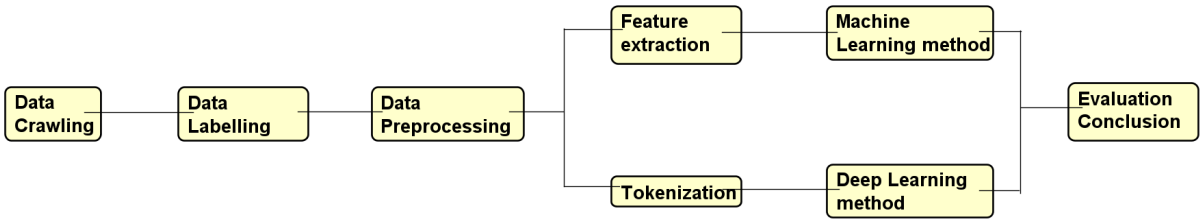


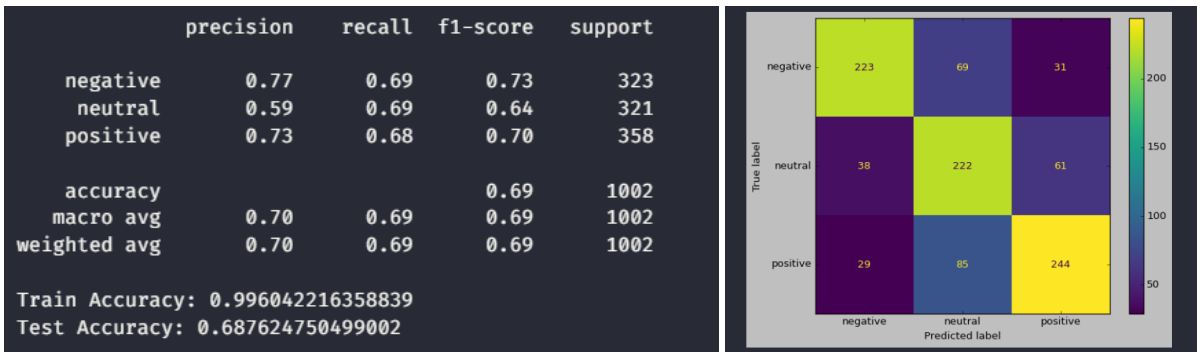
Figure 18: Main Workflow

5.1.1 Dataset Preparation

The dataset is divided into 60% training set and 40% test set (already attached), using stratified sampling to maintain the distribution of positive and negative reviews in both sets. This ensured that our models were trained and evaluated on representative data. Then, we visualized the distribution of ratings and review lengths using histograms and box plots to gain insights into the data's characteristics and identify any potential outliers or data anomalies. A further-detailed description of our dataset is already mentioned in "Data Preprocessing" section.

5.1.2 Evaluation Metrics

We chose accuracy as our evaluation metric for the problem because of its popularity for classification problems, which is calculated as the total number of correct classifications divided by the total number of instances. The confusion matrix is also chosen as it enables us to see which classes are misclassified with which classes. All the evaluations for each models are recorded and saved in each model's theory section. Here is the example of the recorded evaluation:



	precision	recall	f1-score	support
negative	0.77	0.69	0.73	323
neutral	0.59	0.69	0.64	321
positive	0.73	0.68	0.70	358
accuracy			0.69	1002
macro avg	0.70	0.69	0.69	1002
weighted avg	0.70	0.69	0.69	1002

Train Accuracy: 0.996042216358839
Test Accuracy: 0.687624750499002

Figure 19: Accuracy and Confusion Matrix of Random Forest model with TF - IDF techniques

5.1.3 Optimizers and Schedulers

For the optimizer of deep learning model, we used the AdamW optimizer because of its popularity and its ability to get learned parameters of the model to converge quickly during training. For the scheduler, we utilized an exponential learning rate which decreases the learning rate by a constant percentage after each epoch as the model tends to need a smaller learning rate when as it approaches the minimum points.

5.1.4 Training

We utilized PyTorch, Tensorflow and Sklearn as the main tools for us to train and evaluate our models. Besides, Weights & Biases is also the platform for us to log and save the checkpoints. In this subsection, we will describe which strategy we used for training and evaluating and which strategies we used to train for each model.

With TF - IDF combined with Random Forest model, the training and test data are first cleaned by removing unnecessary characters and stopwords. The preprocessed texts are stored along with their corresponding labels. The TF - IDF vectorizer is then applied to convert the preprocessed text data into numerical representations by calculating the importance of each word in a document and across all documents. After that, the Random Forest classifier is trained using the TF - IDF transformed training data and their labels. The trained model is then used to predict labels for the TF - IDF transformed test data.

For DBLSTM and DBGRU models, labels are encoded using one-hot encoding. The TextVectorization layer is used to convert the text inputs into numerical representations after shuffling, batching, and pre-fetching the dataset. The models are adapted to the training data to set the vocabulary. They are then compiled with binary cross-entropy loss, AdamW optimizer, and accuracy as the evaluation metric. The only difference between these two models is the model architecture (LSTM layers and GRU layers).

The PhoBERT model is fine-tuned from a pre-trained one in the Hugging Face Transformers library. It involves defining hyperparameters such as the number of epochs, batch size, learning rate, and warm-up steps. Then, loading the model and tokenizer, tokenizing the input text, setting up the training arguments and metrics, and finally, training the model.

Finally, XLM - RoBERTa model is implemented using the PyTorch framework, which sets hyperparameters such as the number of epochs, batch size, learning rate, and padding values. We utilized the “Torchtext” and “Transformers” libraries for the tokenization on the text data and training arguments. The strategy used for training involves iterating over the dataset, updating the model’s parameters, and monitoring performance metrics.

5.2 Quantitative Results & Evaluations

In this section, we will present the results of each model for each dataset and make comparisons. We will also give some reasons why some models work better than others for the dataset. After applying various techniques and experimenting with different parameter settings, here is the summary of the results of different models with the best-performing parameter settings we have tried.

Model	Train Accuracy (%)	Test Accuracy (%)	Inference Time	Input Fields
PhoBERT	99.86	81.24	7.43h	Text Fields Only
RF + TF - IDF	99.61	68.76	3.2s	Text Fields Only
DBLSTM	99.60	62.28	300.1s	Text Fields Only
DBGRU	99.80	63.67	412.4s	Text Fields Only
XLM - RoBERTa	**99.92**	**82.04**	3.30h	Text Fields Only

Table 1: Classification results for each model, RF stands for Random Forest, DBLSTM and DBGRU stand for Deep Bidirectional Long Short-term Memory and Gated Recurrent Units, respectively.

We can observe the outperformance of the pre-trained BERT model over the Machine Learning and Neural Network models, while Neural Network models performs slightly worse than Random Forest models. First, XLM - RoBERTa model, with a better pre-training strategy, reached quite a better result than the pre-trained PhoBERT model. Specifically, this is surprising that XLM - RoBERTa gave the best results for training on the dataset itself with an accuracy of 82.04%, compared to 81.24% accuracy of PhoBERT, and the much lower inference time of 3.30 hours, compared to the inference time of PhoBERT (7.43 hours). Theoretically, PhoBERT should perform better thanks to its better support for the Vietnamese language but we believed that due to the fact that the process of fine-tuning involves training the pre-trained models on task-specific data. The training regimen, including the choice of

the optimizer, learning rate, batch size, and number of epochs, can impact the models' ability to learn and generalize patterns in sentiment analysis. It is possible that the fine-tuning process for XLM - RoBERTa was more effective in optimizing the model's performance for sentiment analysis compared to PhoBERT. Another reason for the superior performance of XLM-RoBERTa over PhoBERT could be attributed to the differences in their pre-training tasks. PhoBERT was pre-trained using another specific set of Vietnamese datasets from RoBERTa, while XLM-RoBERTa underwent an upgraded pre-training task. The upgraded pre-training task of XLM-RoBERTa might have included additional improvements, techniques, or a larger and more diverse training corpus, leading to enhanced language modeling capabilities and better performance on downstream tasks like sentiment analysis. These differences in pre-training strategies could contribute to the variations in their effectiveness for the specific task at hand. Secondly, with a train accuracy of 99.60%, the Random forest model combined with TF - IDF techniques performs well during training. However, the test accuracy is relatively lower at 68.76%, indicating some difficulty in generalizing to unseen data, compared to BERT models. Last but not least, DBLSTM shows the lowest accuracy rate on both the training and test data. Similar to DBLSTM, DBGRU model achieves a train accuracy of 99.80%. The test accuracy had small improvement to 63.67%, suggesting a slightly better generalization capability. But, the inference time is higher at 412.4 seconds compared to DBLSTM. The reason for these differences in evaluation between these models could be that it is generally language-agnostic and does not incorporate language-specific features explicitly. In contrast, models like XLM - RoBERTa are pre-trained on large-scale language-specific data, enabling them to learn language-specific patterns and improve performance on language-specific tasks. Since the sentiment analysis task involves the Vietnamese language, the language-specific capabilities of XLM - RoBERTa could give it an edge over the Deep Bidirectional Neural Networks.

Additionally, from the classification report of five models [6] [12] [13] [15] [17], it can be inferred that our models achieved reasonably good performance with pretty high precision, recall, and F1-scores for the "positive" and "negative" sentiment classes, while the "neutral" class had slightly lower scores, demonstrates commendable performance in classifying stock article sentiments. They both show strong ability to identify positive and negative sentiments accurately, while also displaying a reasonable level of performance in detecting articles with neutral sentiments, even though the models have an adequate number of neutral samples to learn from (with a support of 321 instances). The slightly lower scores in the neutral class may indicate a greater challenge in identifying and classifying neutral sentiments accurately. Therefore, when the model says a sentiment is neutral, the observes should stay careful. Further analysis and evaluation may be required to identify areas for improvement and fine-tuning in order to enhance the models' performance in classifying neutral sentiments more effectively.

6 Conclusions & Possible Extensions

So far, we have introduced and solved our prediction problem by different techniques such as deep learning models and natural language processing techniques. The model that shows the best classification performance, in general, is the XLM - RoBERTa model which performs the most effectively with both training and testing set on the dataset.

It is the first time we have done a capstone project on such a big scale so it taught us many lessons. Firstly, we had to personally collect and label the text data, which consumed a significant amount of time. Secondly, we faced resource limitations when handling this dataset. Additionally, a thorough understanding of our problem was essential, including clearly defining the input and output components. Finally, data preprocessing and model selection can have a tremendous effect on the result of our solution.

One of the most challenging aspects of solving our problem was determining the appropriate storage method for our dataset, as well as selecting the models and techniques for workload assignments among team members. Ultimately, we opted for the aforementioned techniques and models discussed in earlier sections. And finally, we decided to follow an aforementioned process.

As we can see from the table of results of different models, BERT model, especially XLM - RoBERTa yiled much better results in term of errors for this type of problem. Therefore, given more time on this project, we would have annotated more data, found better training strategy for each models and further explored new AI models such as ViDeBERTa [9], which is considered a state-of-the-art model for Vietnamese, tried the Universal Language Model Fine-tuning techniques, which pre-trains a language model on a large general-domain corpus and fine-tunes it on the target task using novel techniques. Additionally, we would have delved into the application of self-supervised and semi-supervised learning methods on our preprocessed dataset.

All the codes, notebooks, trained weights model files for the project with detailed instructions can be found in the following Github repository: https://github.com/helloitsurdvq/StockReview.IntroAI.20222

7 Contributions

In this section, we will inform the work each group member has done by listing works where he mainly involved based on in this project.

7.1 Programming, Coding

- Idea proposal

  Nguyen Chi Long - 20210553 (60%)
  Dinh Viet Quang - 20215235 (40%)

- Data crawling:

  Dinh Viet Quang - 20215235 (50%)
  Le Xuan Hieu - 20215201 (50%)

- Labelling and create dataset:

  Dinh Viet Quang - 20215235 (100%)

- Exploratory Data Analysis (EDA):

  Ngo Xuan Bach - 20215181 (30%)
  Nguyen Chi Long - 20210553 (70%)

- Data splitting:

  Dinh Viet Quang - 20215235 (100%)

- TF - IDF with Random Forest model:

  Ngo Xuan Bach - 20215181

- DBGRU model:

  Ngo Xuan Bach - 20215181

- DBLSTM model:

  Le Xuan Hieu - 20215201

- PhoBERT:

  Nguyen Chi Long - 20210553

- XLM - RoBERTa:

  Dinh Viet Quang - 20215235

- Demonstration

  Nguyen Chi Long - 20210553 (50%)
  Dinh Viet Quang - 20215235 (50%)

7.2 Report Writing, Readme file

- Report Writing:

  Nguyen Chi Long - 20210553 (20%)
  Ngo Xuan Bach - 20215181 (30%)
  Le Xuan Hieu - 20215201 (20%)
  Dinh Viet Quang - 20215235 (30%)

- Readme file:

  Dinh Viet Quang - 20215235 (100%)

Demonstration and Deployment

We created a notebook to demonstrate the performance of the models in our project (DBLSTM, DBGRU, PhoBERT, XLM - RoBERTa) by entering a sentence or all the content of an article, after that, it will display a pie chart showing the percentage of sentiment labels in the content. We highly recommend using the Google Colab version along with setting a suitable model path because of its built-in libraries and our models are quite large.

Figure 20: Prediction of sentence content and DBGRU model in notebook

We have also implemented the web deployment of our project using Flask. To begin using our user interface tool, please review and follow the provided instructions in “Readme.md” file carefully. An illustrative example of our application can be seen in the below figure.

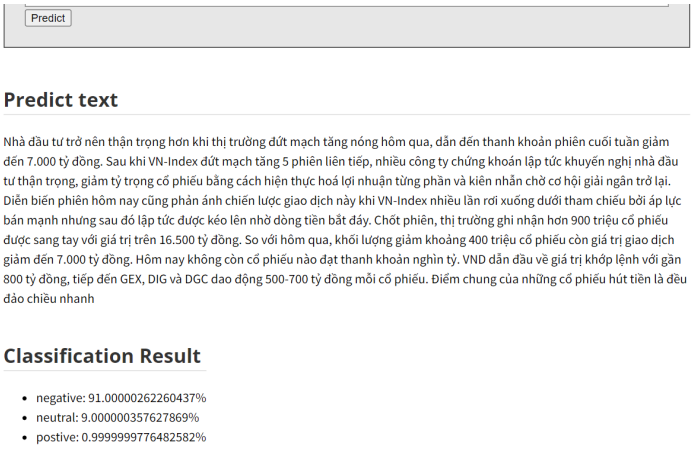


Figure 21: Prediction of VnExpress article content and XLM - RoBERTa model in web application

References

[1] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online, July 2020. Association for Computational Linguistics.

[2] Mamata Das, Selvakumar Kamalanathan, and PJA Alphonse. A comparative study on tf-idf feature weighting method and its analysis using unstructured dataset. In *COLINS*, pages 98–107, 2021.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

[4] Philip Gage. A new algorithm for data compression. *The C Users Journal archive*, 12:23–38, 1994.

[5] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052 vol. 4, 2005.

[6] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[7] Dat Quoc Nguyen and Anh Tuan Nguyen. Phobert: Pre-trained language models for vietnamese. *arXiv preprint arXiv:2003.00744*, 2020.

[8] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[9] Cong Dao Tran, Nhut Huy Pham, Anh Nguyen, Truong Son Hy, and Tu Vu. Videberta: A powerful pre-trained language model for vietnamese, 2023.

[10] Trieu H. Trinh and Quoc V. Le. A simple method for commonsense reasoning. *CoRR*, abs/1806.02847, 2018.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[12] Thanh Vu, Dat Quoc Nguyen, Dai Quoc Nguyen, Mark Dras, and Mark Johnson. VnCoreNLP: A Vietnamese natural language processing toolkit. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 56–60, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[13] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, 2015.