VIETNAMESE SENTIMENT ANALYSIS

Nguyễn Mạnh Đan - 2052932

1. Introduction

Sentiment analysis is a natural language processing technique used to determine whether data is positive, negative, or neutral. Sentiment analysis is often performed on textual data to help businesses monitor brand and product sentiment in customer feedback, and understand customer needs.

To perform sentiment analysis on a complex language such as Vietnamese, our model would need to employ a suitable cleaning pipeline as well as a sophisticated model that can capture the meaning of the dataset.

We believe that this project will help us refresh, understand more, and validate the knowledge we have learnt during the course and improve our practical insight on how to preprocess the data and train the models.

2. Solutions

Traditionally, sentiment analysis was done using machine learning methods such as Support Vector Machine (SVM), Random Forest, Naive Bayes,… [1]. Those methods had achieved acceptable results and are still used nowadays in the situations where the dataset is small, the computational power is limited or the interpretability of the model is crucial.

Modern approach to sentiment analysis employ the power of deep learning models built upon the Convolution Neural Networks, Recurrent Neural Networks or Transformer architectures. Currently, state-of-the-art models like XLNet of EFL have achieved top performance of more than 96% accuracy on the IMDb movie review dataset [2].

Concerning the Vietnamese language, much progress has been done to advance the sentiment analysis of Vietnamese texts [3]. On Vietnamese text datasets such as VLSP or AIVIVN, both machine and deep learning methods were used and have achieved good results. Therefore, we will experiment with both machine and deep learning approach to determine which is more effective for this project.

3. Approach

In this project, we will experiment with both machine learning approach and deep learning approache. The machine learning approach consists of Support Vector Machine, Random Forest, and Naive Bayes methods. The deep learning approach consists of Convolution Neural Networks, Long Short-term Memory Neural Networks, and Convolution Long Short-term Memory Neural Networks methods.

The the implementation code can be found at my Github repository.

3.1. Machine learning approach

3.1.1. Data preprocessing

The dataset of this experiment is the VLSP dataset which is a collections of Vietnamese texts, each text has a label indicating the sentiment that text. The training set contains 5100 rows and the testing set contains 1050 rows.

For the labels, we left them be as the default values already indicate which sentiment class the text is (1 for positive, 0 for neutral, and -1 for negative).

For the texts, we implemented a cleaning pipeline which consists of the following steps:

- Removing HTML elements in the text if available.
- Converting Windows-1252 characters to Unicode UTF-8 characters.
- Normalizing tones such as "òa" to "oà" so that words like "hoà" and "hoài" have the same "oà" spelling.
- Removing special characters such as punctuations, numbers, and two or more spaces from the texts.

Additionally, the removal of stopwords from the text had also been tested but the final results showed that such action only lowered the model accuracy so we did not implement that feature

3.1.2. Input vectorization

a/ TF-IDF

To convert the input texts to vectors, first we calculated the Term Frequency, the number of times a word appear in a sentence, and the Inverse Document Frequency, the inverse of the number of times a word appear in all sentences, then divide the Term Frequency by the Inverse Document Frequency to get the TF-IDF representation of a sentence.

b/ Word2Vec

In addition to TF-IDF, we also experimented with 2 word embedding models. The first model was the Word2Vec pre-trained on Vietnamese texts. This model took in each word in our vocabulary and returned a vector of size 400. In case the word in our vocabulary was not in the model, we substituted that word vector of size 400 with 0 values. Each word in a sentence is a vector of size 400, to represent a sentence, we computed the average vector by summing up all the word vectors then divide by the number of words.

c/ SBERT

Alternatively, we also experimented with an SBERT model pre-trained on Vietnamese texts to see how the model performed on another word embedding model. This model took in each sentence and returns a vector of size 768.

3.1.3. Model architecture & training

a/ With grid search

For the TF-IDF input vectorization method, we began with a base model then perform grid search with a set of parameters for each model to determine the best performing model on the training set.

The parameter set concerning Support Vector Machine are:

- The values of C: A regularization parameter that controls the trade-off between achieving a low training error and a low testing error. The values we experimented with are [0.1, 1, 10, 100].
- Kernel: The kernel function mapping the input features into a higher-dimensional space. The values we experimented with are ["linear", "poly", "rbf", "sigmoid"].

The best performing SVM model has the C of 1 and the kernel of "rbf".

The parameter set concerning Random Forest are:

- n_estimators: The number of trees in the Random Forest. The values we experimented with are [50, 100, 150].
- max_depth: The maximum depth allowed for each tree in the Random Forest. The values we experimented with are [None, 10, 20].
- min_samples_split: The minimum number of samples required to split an internal node. The values we experimented with are [2, 5, 10].
- min_samples_leaf: The minimum number of samples required to be at a leaf node. The values we experimented with are [1, 2, 4].

The best performing Random Forest model has n_estimators of 100, the max_depth of None, the min_samples_split of 5, and the min_samples_leaf of 1.

The parameter set concerning Naive Bayes is:

- The values of alpha: A smoothing parameter to handle the issue of zero probabilities for certain events in the training data. The values we experimented with are [0.1, 0.5, 1, 2].

The best performing Naive Bayes model has the alpha of 2.

b/ Without grid search

For the W2V and SBERT input vectorization method, the grid search took a long time to finish so we only performed SVM, Random Fores with the default parameter values. The Naive Bayes method cannot be used with the W2V and SBERT input vectorization method because the input vectors contain negative values.

3.2. Deep learning approach

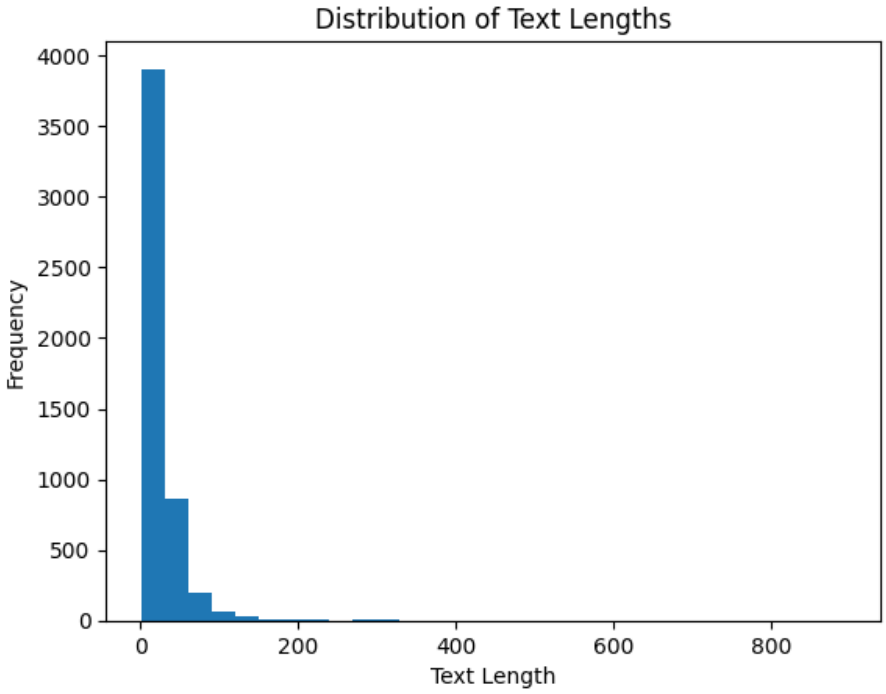
3.2.1. Data preprocessing

The data preprocessing part of the deep learning approach is mostly the same as that of the machine learning approach, but with some certain differences.

For the labels, we encoded with one-hot encoding to use the softmax activation function in the output layer. Since there are 3 classes, we encoded 1 with [1, 0, 0], 0 with [0, 1, 0], and -1 with [0, 0, 1].

After the texts were cleaned, we tokenized and converted the texts into sequences of the same length before computing the word vector. To determine the length of each sequence, we took into account 2 factors.

First was the frequency of the length of each sentence. According to this graph generated from the dataset, most sentences tend to be shorter than 400 characters.



Second was the max text length specified by the pre-trained word embedding model. In the case of SBERT, which is one of the word embedding methods, the max text length is 256.

Combining two factors, we decided that 256 would be the length of each sequence.

3.2.2. Word embedding

We also used W2V and SBERT as the embedding model for the deep learning approach. The input sentences with be converted to embedding vectors, the collection of the embedding vectors will form an embedding matrix which served as the embedding layer of our models.

3.2.3. Model architecture

a/ Convolutional Neural Networks

```
Model: "cnn_w2v"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_3 (InputLayer)        [(None, 256)]             0         []

 embedding_2 (Embedding)     (None, 256, 400)          3160400   ['input_3[0][0]']

 conv1d_6 (Conv1D)           (None, 254, 100)          120100    ['embedding_2[0][0]']

 conv1d_7 (Conv1D)           (None, 253, 100)          160100    ['embedding_2[0][0]']

 conv1d_8 (Conv1D)           (None, 252, 100)          200100    ['embedding_2[0][0]']

 max_pooling1d_6 (MaxPoolin  (None, 1, 100)            0         ['conv1d_6[0][0]']
 g1D)

 max_pooling1d_7 (MaxPoolin  (None, 1, 100)            0         ['conv1d_7[0][0]']
 g1D)

 max_pooling1d_8 (MaxPoolin  (None, 1, 100)            0         ['conv1d_8[0][0]']
 g1D)

 concatenate_4 (Concatenate  (None, 3, 100)            0         ['max_pooling1d_6[0][0]',
 )                                                                 'max_pooling1d_7[0][0]',
                                                                   'max_pooling1d_8[0][0]']

 attention_2 (Attention)     (None, 3, 100)            0         ['concatenate_4[0][0]',
                                                                   'concatenate_4[0][0]']

 concatenate_5 (Concatenate  (None, 3, 200)            0         ['concatenate_4[0][0]',
 )                                                                 'attention_2[0][0]']

 flatten (Flatten)           (None, 600)               0         ['concatenate_5[0][0]']

 dropout_2 (Dropout)         (None, 600)               0         ['flatten[0][0]']

 dense_4 (Dense)             (None, 128)               76928     ['dropout_2[0][0]']

 dense_5 (Dense)             (None, 3)                 387       ['dense_4[0][0]']

==================================================================================================
Total params: 3718015 (14.18 MB)
Trainable params: 3718015 (14.18 MB)
Non-trainable params: 0 (0.00 Byte)
```

The overall architecture of the CNN is as followed:

- 3 conv layers, each using the ReLU activation function and containing 100 filters of size 3x3, 4x4, and 5x5 respectively.
- 3 1D max-pooling layers of stride 1 and window sizes of 254, 253, and 252 respectively.

- 1 output layer of 3 classes using the softmax activation function.
- L2 regularization with lambda being 0.01 and a dropout rate of 0.5 to prevent overfitting.
- Adam optimizer with a learning rate of 0.001

Additionally, we also made a few changes that would improve the model performance:

- Adding an attention layer after the max-pooling layers, a key mechanism of the Transformers architecture, to allow the model to dynamically weigh different words or phrases in the input text, giving more attention to those that are likely to be more indicative of the sentiment expressed.
- Adding another dense layer of size 128 with the ReLU activation layer right before the output layer.

b/ Long Short-term Memory Neural Networks

```
Model: "lstm_w2v"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_4 (InputLayer)        [(None, 256)]             0         []

 embedding_3 (Embedding)     (None, 256, 400)          3160400   ['input_4[0][0]']

 lstm_2 (LSTM)               (None, 128)               270848    ['embedding_3[0][0]']

 attention_3 (Attention)     (None, 128)               0         ['lstm_2[0][0]',
                                                                  'lstm_2[0][0]']

 concatenate_6 (Concatenate  (None, 256)               0         ['lstm_2[0][0]',
 )                                                                'attention_3[0][0]']

 dropout_3 (Dropout)         (None, 256)               0         ['concatenate_6[0][0]']

 dense_6 (Dense)             (None, 128)               32896     ['dropout_3[0][0]']

 dense_7 (Dense)             (None, 3)                 387       ['dense_6[0][0]']

=================================================================================================
Total params: 3464531 (13.22 MB)
Trainable params: 3464531 (13.22 MB)
Non-trainable params: 0 (0.00 Byte)
```

The overall architecture of the LSTM was also the same as the LSTM in lab 6:

- 1 LSTM layer of size 128. In our experiment, we concluded that more LSTM layers or bigger LSTM sizes did little to improve the model accuracy but affected the training time a lot. Therefore we only used 1 LSTM layer of size 128.
- 1 output layer of 3 classes using the softmax activation function.
- L2 regularization with lambda being 0.01 and a dropout rate of 0.5 to prevent overfitting.
- Adam optimizer with a learning rate of 0.001

Additionally, we also added an attention layer after the LSTM layer and another dense layer before the output layer.

c/ Convolutional Long Short-term Memory Neural Networks

```
Model: "cnn_lstm_w2v"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_5 (InputLayer)        [(None, 256)]             0         []

 embedding_4 (Embedding)     (None, 256, 400)          3160400   ['input_5[0][0]']

 reshape_2 (Reshape)         (None, 256, 400)          0         ['embedding_4[0][0]']

 conv1d_9 (Conv1D)           (None, 254, 100)          120100    ['reshape_2[0][0]']

 conv1d_10 (Conv1D)          (None, 253, 100)          160100    ['reshape_2[0][0]']

 conv1d_11 (Conv1D)          (None, 252, 100)          200100    ['reshape_2[0][0]']

 max_pooling1d_9 (MaxPoolin  (None, 1, 100)            0         ['conv1d_9[0][0]']
 g1D)

 max_pooling1d_10 (MaxPooli  (None, 1, 100)            0         ['conv1d_10[0][0]']
 ng1D)

 max_pooling1d_11 (MaxPooli  (None, 1, 100)            0         ['conv1d_11[0][0]']
 ng1D)

 concatenate_7 (Concatenate  (None, 3, 100)            0         ['max_pooling1d_9[0][0]',
 )                                                                 'max_pooling1d_10[0][0]',
                                                                   'max_pooling1d_11[0][0]']

 lstm_3 (LSTM)               (None, 128)               117248    ['concatenate_7[0][0]']

 attention_4 (Attention)     (None, 128)               0         ['lstm_3[0][0]',
                                                                   'lstm_3[0][0]']

 concatenate_8 (Concatenate  (None, 256)               0         ['lstm_3[0][0]',
 )                                                                 'attention_4[0][0]']

 dropout_4 (Dropout)         (None, 256)               0         ['concatenate_8[0][0]']

 dense_8 (Dense)             (None, 128)               32896     ['dropout_4[0][0]']

 dense_9 (Dense)             (None, 3)                 387       ['dense_8[0][0]']

=================================================================================================
Total params: 3791231 (14.46 MB)
Trainable params: 3791231 (14.46 MB)
Non-trainable params: 0 (0.00 Byte)
```

The overall architecture of the CNN_LSTM was the combination of the above CNN and LSTM, with an addition of the attention layer and another dense layer.

3.2.4. Model training

To set up the training loop, first, we defined the early stopping mechanism monitoring the val_accuracy with the min_delta of 0.01 and patience of 10. We also set True to restore_best_weights so that the final model would be the one that has the lowest val_loss in all epochs.

Then we randomly split the training data into training and validation sets with a ratio of 8:2. The model would be trained on the training set and be cross-validated with the validation set.

Changing between 2 word embedding models and 3 deep learning architectures, we trained 6 of our models with 30 epochs and a batch size of 256.

4. Comparisons

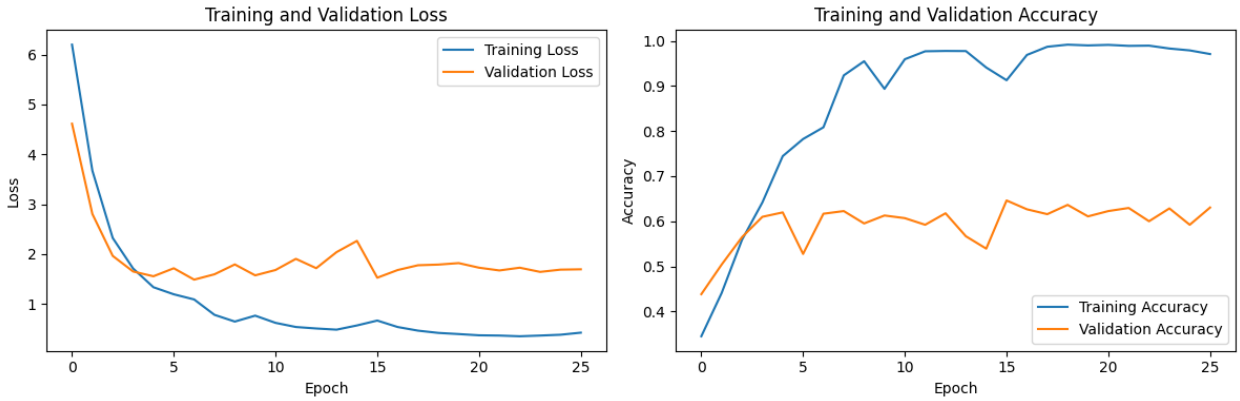
	Embedding	**Loss**	**Accuracy**	**Training**	**Params**	**Note**
SVM	TF-IDF	-	**70.00%**	548.07s	-	Grid Search
RF	TF-IDF	-	64.00%	346.81s	-	Grid Search
NB	TF-IDF	-	67.00%	1.26s	-	Grid Search
SVM	W2V	-	63.00%	8.59s	-	
RF	W2V	-	55.00%	11.49s	-	
NB	W2V	-	-	-	-	Negative values
SVM	SBERT	-	**69.00%**	8.11s	-	
RF	SBERT	-	64.00%	14.87s	-	
NB	SBERT	-	-	-	-	Negative values
CNN	W2V	1.98	67.33%	59.19s	3718015	Early Stopping
LSTM	W2V	1.02	68.95%	37.11s	3464531	Early Stopping
CNN_LSTM	W2V	1.09	**74.76%**	62.24	3791231	Early Stopping
CNN	SBERT	1.20	65.14%	94.41s	4916015	Early Stopping
LSTM	SBERT	0.99	66.28%	48.15s	4409347	Early Stopping
CNN_LSTM	SBERT	1.49	62.66%	68.84s	4989231	Early Stopping

All the machine learning model training and testing were performed on Google Colab with CPU. All the deep leaning model training and testing were performed on Google Colab with GPU T4.

Regarding the machine learning methods, the SVM architecture combined with the TF-IDF feature extraction and grid search had the highest accuracy of 70%, followed closely by SVM with SBERT embedding with the accuracy of 69%. This suggests that SVM is the top performing architecture for the machine learning approach and TF-IDF and SBERT are effective in converting text input into vectors. However, the training time of SVM when including grid

search increased significantly since the machine learning approach were trained on CPU, making it impractical to train the W2V and SBERT models with grid search.

Regarding the deep learning methods, the CNN and LSTM hybrid model by far achieved the highest accuracy of 74.76%, making it the highest accuracy model overall.



CNN-LSTM_W2V Loss and Accuracy

All deep learning models utilized early stopping to save resources and prevent overfitting. This makes the training time of each model vary but most stopped at between the 15th and the 25th epoch, making the overall training time of the deep learning methods short that that of the machine learning methods with grid search.

We believe that the deep learning model combining CNN and LSTM is able to handle the complexity of sequential data like texts thanks to the RNN architecture better than the machine learning models like SVM which leads to its ability to string together words more effectively and therefore can more correctly classify the sentiment of the input texts.

5. Conclusion

This project had helped us gain more insight into the architecture of machine learning and deep learning models as well as how to better preprocess data and train the models. As sentiment analysis is a complex task and Vietnamese is a complex language, to improve the performance further, we plan to experiment with more word embedding models and other architecture such as Transformer. Additionally, a more complex preprocessing pipeline that takes into abbreviations could also be implemented to improve performance.

6. References

[1] El Mahdi Mercha & Houda Benbrahim (2023), Machine learning and deep learning for sentiment analysis across languages: A survey, https://doi.org/10.1016/j.neucom.2023.02.015.
[2] Sentiment Analysis on IMDb. paperswithcode.
https://paperswithcode.com/sota/sentiment-analysis-on-imdb

[3] NLP-Vietnamese-progress. undertheseanlp.
https://github.com/undertheseanlp/NLP-Vietnamese-progress