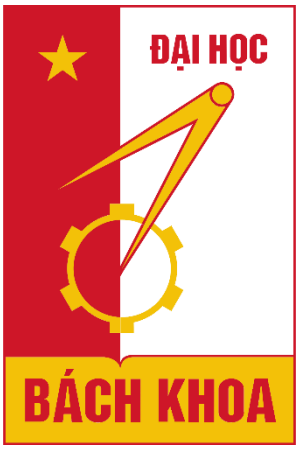
Đại học Bách Khoa Hà Nội

Viện Công nghệ thông tin và Truyền thông



Báo cáo Project II

Phân loại sắc thái bình luận trong phản hồi của khách hàng

Giảng viên hướng dẫn:

TS. Đỗ Bá Lâm

Họ tên sinh viên: Dương Minh Hoàng

MSSV: 20183541

Lớp: Khoa học máy tính 04 – K63

Hà Nội – ngày 21 tháng 5 năm 2021

Lời cảm ơn

Lời đầu tiên, em xin được gửi lời cảm ơn đến thầy Đỗ Bá Lâm - giảng viên hướng dẫn môn Đồ án 2 (Project II) , Viện Công nghệ thông tin và Truyền thông, đại học bách khoa Hà Nội đã tận tâm nhiệt tình chỉ dạy, truyền đạt những kiến thức vô cùng quý giá đối với em trong suốt kì học vừa qua tại trường. Những kiến thức, kinh nghiệm em tiếp thu được là cơ sở quan trọng nhất giúp em hoàn thành đồ án môn học và sẽ là hành trang quý giá để em tiếp tục trên con đường sắp tới, tiếp tục hoàn thiện bản thân hơn trong sự nghiệp cũng như cuộc sống.

Trong quá trình thực hiện đồ án, em đã cố gắng và nỗ lực hết mình nhưng do hạn chế về mặt thời gian, kiến thức và cơ sở vật chất nên đồ án không thể tránh khỏi những sai sót. Em mong thầy cùng các bạn có thể chỉ bảo, đóng góp để hoàn thiện hơn.

Em xin chân thành cảm ơn!

Nghệ An, ngày 9 tháng 6 năm 2021

Mục lục:

1. Lời mở đầu

Phản hồi của khách hàng là những ý kiến của người dùng đưa ra cho doanh nghiệp theo nhiều tính chất khác nhau. Chúng có thể mang tính xây dựng sản phẩm, mong muốn của khách hàng về sản phẩm, dịch vụ mà họ được cung cấp. Nó cũng thể hiện sự đánh giá của khách về mức độ hài lòng, đưa ra những gì họ thích hoặc không thích về sản phẩm, công ty. Phản hồi còn có thể là những khiếu nại về các vấn đề khách hàng gặp phải khi sử dụng sản phẩm của công ty.

Sự phản hồi của khách hàng rất quan trọng đối với mỗi doanh nghiệp vì nó phản ánh khách quan về chất lượng của sản phẩm, giúp cho công ty có định hướng cải thiện và phát triển. Đồng thời, đây còn là công cụ quảng cáo, thúc đẩy cho doanh nghiệp trong quá trình kinh doanh.

Thương mại điện tử có nhiều ưu điểm so với thương mại truyền thống. Bên kinh doanh không cần mặt bằng, bên khách hàng có thể xem và đặt hàng từ nhà. Một trong những yếu tố quyết định một mặt hàng bán chạy hay không là đánh giá của những khách hàng đã mua trước đó. Một khách hàng mới có thể dựa trên các đánh giá đó trước khi đưa ra quyết định. Nhà sản xuất cũng có thể dựa vào các đánh giá đó để cải thiện chất lượng sản phẩm. Việc phân loại các đánh giá theo hai mức độ tích cực và tiêu cực đóng vai trò quan trọng trong việc cải thiện chất lượng sản phẩm và xây dựng các hệ thống gợi ý. Trên cơ sở đó, cuộc thi Sentiment Analysis của AIViVN vào năm 2019 được xây dựng để chúng ta có thể dựa trên một bộ cơ sở dữ liệu về các đánh giá tiêu cực và tích cực để dự đoán sắc thái của các đánh giá mới.

2. Tổng quan về bài toán Sentiment Analysis

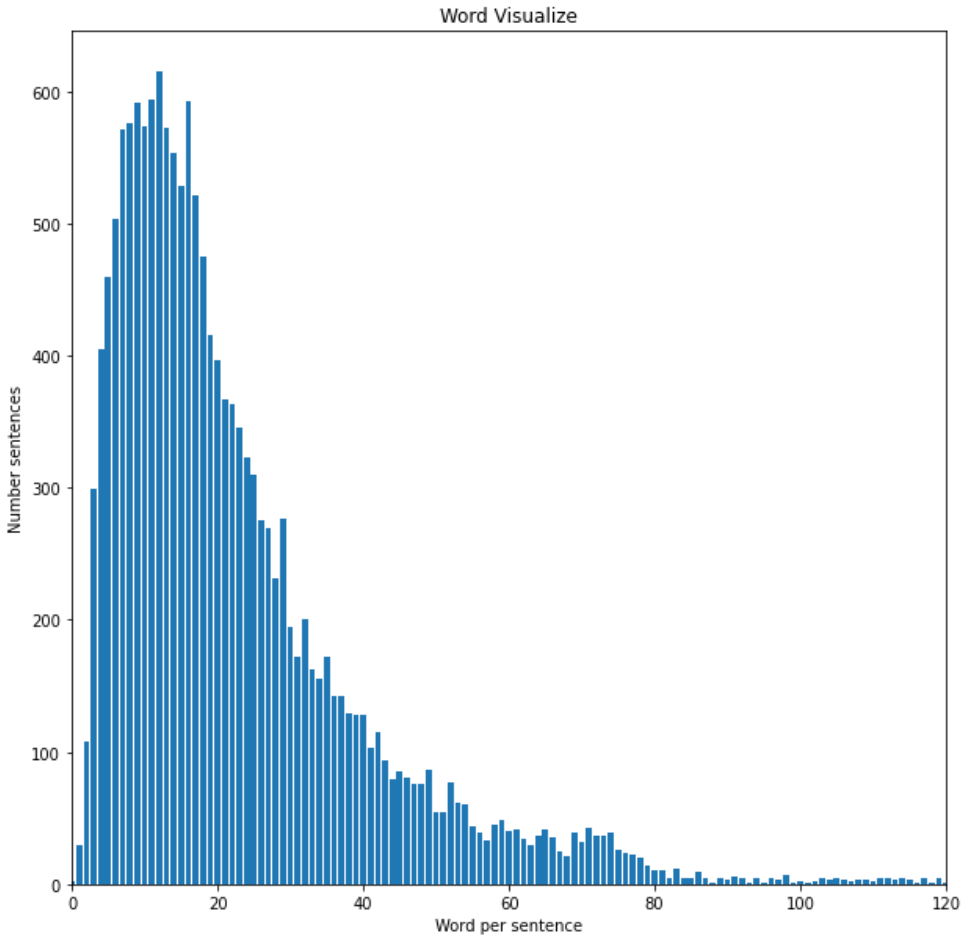
2.1 Bộ dữ liệu

Mỗi mẫu dữ liệu bao gồm 3 thuộc tính chính như sau:

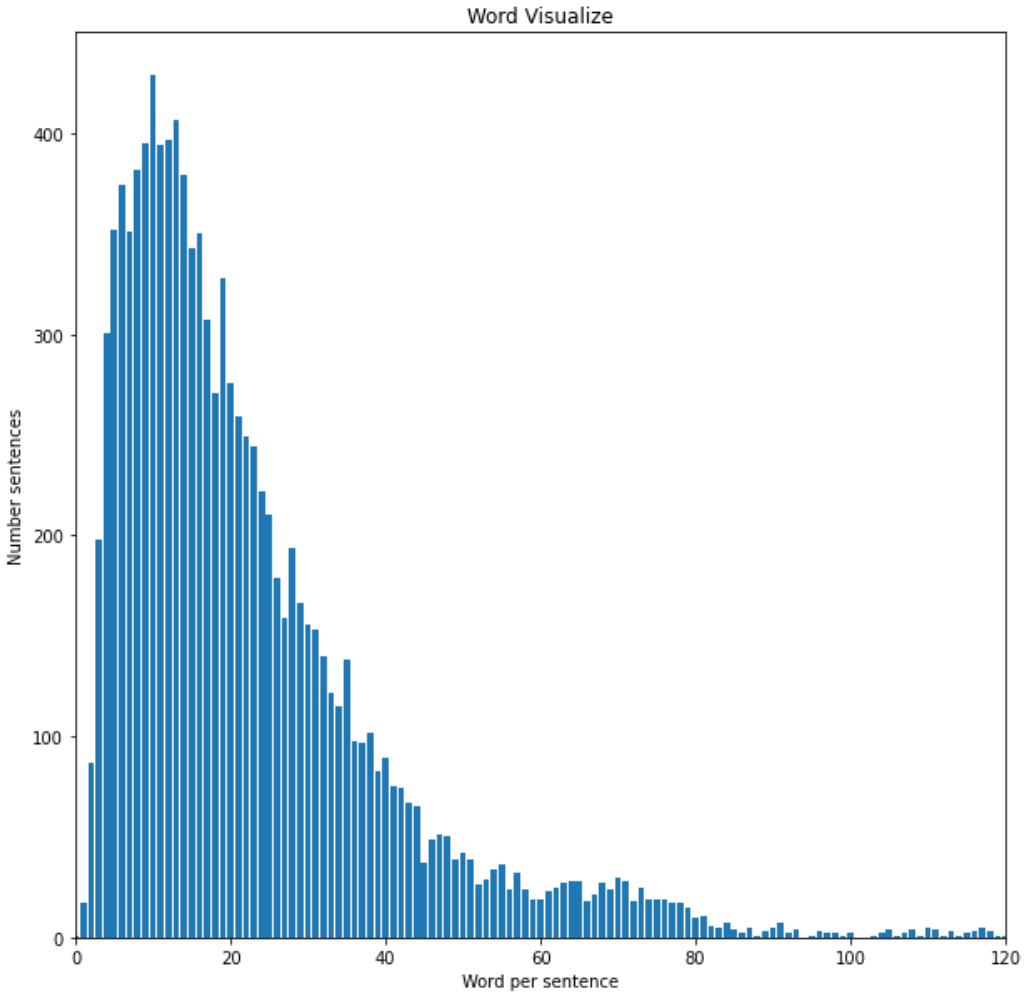
- ID: id duy nhất cho mỗi phản hồi của khách hàng trong tập dữ liệu.
- Content: Nội dung phản hồi của khách hàng.
- Label: Nhãn, thể hiện phản hồi là tích cực hay tiêu cực
  - 0: Tích cực
  - 1: Tiêu cực

Với tính chất của một cuộc thi về học máy, cuộc thi cung cấp 2 file "train.crash" và "test.crash" với tổng cộng 16087 mẫu cho training và 10981 mẫu cho testing cho Public Phase.

2.2 Một số thống kê khác



Hình 1. World Visualize ("Train.crash" File)



Hình 2. World Visualize ("Test.crash" File)

2.3 Nhiệm vụ

Xác định một phản hồi của khách hàng (dưới dạng văn bản) là tích cực hay tiêu cực.

Do đó bài toán được quy về phân loại văn bản dưới dạng 2 nhãn 0 và 1 với 0 là nhãn dành cho phản hồi tích cực và 1 là nhãn dành cho phản hồi tiêu cực.

2.4 Đánh giá

Người tham gia phải gửi tệp .zip có chứa results.csv. Tệp results.csv phải chứa kết quả theo thứ tự như bộ thử nghiệm ở định dạng sau:

id1	xác suất nhãn 1
id2	xác suất nhãn 2
id3	xác suất nhãn 3
…	…

Bài nạp gửi sẽ được đánh giá bằng chỉ số *F1-score* tính theo nhãn tiêu cực.

3. Đề xuất giải pháp

Với việc được quy về dưới dạng bài toán phân loại văn bản với các nhãn 0 và 1 thì những thuật toán phân loại trong học máy quen thuộc sẽ được áp dụng vào bài toán là SVM, Naïve Bayes và Logistic Regression.

Long Short Term Memory networks (LSTM) được giới thiệu bởi Hochreiter & Schmidhuber (1997), và sau đó đã được cải tiến và phổ biến bởi rất nhiều người trong ngành. Chúng hoạt động cực kì hiệu quả trên nhiều bài toán khác nhau nên dần đã trở nên phổ biến như hiện nay. Theo đó LSTM sẽ là mô hình học sâu đầu tiên được lựa chọn để thử nghiệm trong bài toán này.

Bidirectional Encoder Representations from Transformers (BERT) là một mô hình học máy xử lý ngôn ngữ tự nhiên do Google phát triển. BERT được tạo và xuất bản vào năm 2018 bởi Jacob Devlin và các đồng nghiệp của ông từ Google. Nó có thể được sử dụng trong nhiều bài toán NLP như:

- Phân loại văn bản
- Question answering
- Dịch máy
- Nhận dạng thực thể có tên
- Tóm tắt văn bản
- …

Không nghi ngờ gì nữa, *BERT* là một bước đột phá lớn trong việc sử dụng học máy trong xử lý ngôn ngữ tự nhiên. Hơn thế nữa, nó là mã nguồn mở và cho phép tinh chỉnh nhanh chóng, phạm vi ứng dụng thực tế là rất lớn.

Do đó cách tiếp cận tiếp theo được đề ra trong Project II này sẽ được xây dựng xung quanh cốt lõi là một mô hình pretrained transformer, mà cụ thể là mô hình *PhoBERT* được nhóm kĩ sư của VinAI phát triển dành riêng cho Tiếng Việt.

4. Triển khai

4.1. Tiền xử lý dữ liệu

Vì tính chất đây là một cuộc thi nên các nhãn của File "Test.crash" không được public vậy nên việc huấn luyện và kiểm thử dữ liệu đều được tiến hành trên File "Train.crash".

4.1.1 Xử lý dữ liệu cho thuật toán học máy và LSTM

Với việc có khá là nhiều nhiễu cũng như là nhiều kí hiệu đặc biệt trong tập dữ liệu vậy nên để có thể đạt được kết quả tốt trong việc áp dụng thuật toán học máy và LSTM thì dữ liệu cần được xử lý một cách tỉ mỉ

Vì dữ liệu (bình luận của người dùng) ở dạng văn nói nên người dùng thường không quan tâm đến chữ hoa và chữ thường khi gõ, vậy nên chúng ta đưa hết các chữ cái về dạng chữ thường.

Loại bỏ những kí tự kéo dài, ví dụ như: Áo đẹp quáaaaaaaa => Áo đẹp quá

Tiếng Việt có 2 cách bỏ dấu nên đưa về một dạng chuẩn. Ví dụ, chữ "Hòa" và "Hoà" đều được chấp nhận trong Tiếng Việt. Ngoài ra còn một số trường hợp lỗi font chữ và dính chữ cũng cần được chuẩn hóa lại.

Chuẩn hóa một số sentiment word: "okie" => "ok", "okey" => "ok",…

Emoji quy về 2 loại: emojis mang ý nghĩa tích cực (positive): '💯','❤' và emojis mang ý nghĩa tiêu cực (nagative): '👎','😖' .

Người dùng đánh giá 1,2 sao (*) quy hết về 1star, trên 3 sao quy về 5star.

Loại bỏ dấu câu (punctuations) và các ký tự nhiễu.

Xử lý vấn đề phủ định, TF-IDF không xử lý được vấn đề phủ định trong bài toán sentiment. Ví dụ: *Cái áo này rất đẹp* và *Cái áo này chẳng đẹp* sẽ không khác nhau nhiều khi chọn feature tf-idf, giải pháp của mình là biến *chẳng đẹp* thành *not-positive*, hay *không tệ* thành *not-nagative* bằng cách dùng từ điển tâm lý và từ điển phủ định. Từ điển tâm lý được lấy từ VietSentwordnet 1.0 chỉ lấy những từ có score >0.5 bổ sung 1 số sentiment words đặc thù từ tập train.

Augmentation data bằng cách thêm vào các sample của chính tập train nhưng không dấu. (Bình luận không dấu khá phổ biến).(Gấp đôi bộ dữ liệu: bộ dữ liệu mới có 32172 câu).

Việc phân chia tập dữ liệu huấn luyện và kiểm thử đều được thực hiện trên bộ dữ liệu của file train.crash đã được augmentation như trên với tỉ lệ 70:30 từ đó thu được tập dữ liệu huấn luyện (train) có khoảng 22520 câu và tập kiểm thử (valid) có 9652 câu.

Sau đó em sẽ mã hóa dữ liệu theo dạng one-hot với 2 phương pháp chính là TF-IDF Vectors và Count-based Vector.

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer


# create a count vectorizer object
count_vect = CountVectorizer(analyzer='word', token_pattern=r'\w{1,}')
count_vect.fit(X_train)

# transform the training and validation data using count vectorizer object
X_train_counter = count_vect.transform(X_train)
X_test_counter = count_vect.transform(X_test)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

# word level - we choose max number of words equal to 30000 except all words (100k+ words)
tfidf_vect = TfidfVectorizer(analyzer='word', max_features=30000)
tfidf_vect.fit(X_train) # learn vocabulary and idf from training set
X_data_tfidf =  tfidf_vect.transform(X_train)
X_test_tfidf =  tfidf_vect.transform(X_test)
```

4.1.2. Xử lý dữ liệu cho mô hình BERT

a, Về phần tiền xử lý dữ liệu cho BERT thì không cần phải xử lý chi tiết như phần xử lý dữ liệu cho mô hình học máy và LSTM mà chỉ cần làm các bước chính như sau:

- Tách từ bằng thư viện vncorenlp.
- Augmentation bộ dữ liệu bằng cách thêm vào các sample của chính tập train nhưng không dấu (thu được bộ dữ liệu mới có 32172 câu).
- Tập huấn luyện và kiểm thử vẫn chỉ được lấy từ bộ dữ liệu 32172 câu và được phân chia theo tỉ lệ 90:10.

b, Tokenize và padding với kích thước cố định 125

Thuật toán BPE:

- BPE (Byte Pair Encoding) là một kỹ thuật nén từ cơ bản giúp chúng ta index được toàn bộ các từ kể cả trường hợp từ mở (không xuất hiện trong từ điển) nhờ mã hóa các từ bằng chuỗi các từ phụ (subwords). Nguyên lý hoạt động của BPE dựa trên phân tích trực quan rằng hầu hết các từ đều có thể phân tích thành các thành phần con.
- Chẳng hạn như từ: low, lower, lowest đều là hợp thành bởi low và những đuôi phụ er, est. Những đuôi này rất thường xuyên xuất hiện ở các từ. Như vậy khi biểu diễn từ lower chúng ta có thể mã hóa chúng thành hai thành phần từ phụ (subwords) tách biệt là low và er. Theo cách biểu diễn này sẽ không phát sinh thêm một index mới cho từ lower và đồng thời tìm được mối liên hệ giữa lower, lowest và low nhờ có chung thành phần từ phụ là low.
- Phương pháp BPE sẽ thống kê tần suất xuất hiện của các từ phụ cùng nhau và tìm cách gộp chúng lại nếu tần suất xuất hiện của chúng là lớn nhất. Cứ tiếp tục quá trình gộp từ phụ cho tới khi không tồn tại các subword để gộp nữa, ta sẽ thu được tập subwords cho toàn bộ bộ văn bản mà mọi từ đều có thể biểu diễn được thông qua subwords.
- Quá trình này gồm các bước như sau:
    - Bước 1: Khởi tạo từ điển (vocabulary).
    - Bước 2: Biểu diễn mỗi từ trong bộ văn bản bằng kết hợp của các ký tự với token <\w> ở cuối cùng đánh dấu kết thúc một từ (lý do thêm token sẽ được giải thích bên dưới).
    - Bước 3: Thống kê tần suất xuất hiện theo cặp của toàn bộ token trong từ điển.
    - Bước 4: Gộp các cặp có tần suất xuất hiện lớn nhất để tạo thành một n-gram theo level character mới cho từ điển.
    - Bước 5: Lặp lại bước 3 và bước 4 cho tới khi số bước triển khai merge đạt đỉnh hoặc kích thước kỳ vọng của từ điển đạt được.

Bước tiếp theo của bài toán là từ dữ liệu thô ở trên, em sử dụng `bpe` đã load được để đưa text đầu vào dưới dạng subword và ánh xạ các subword này về dạng index trong từ điển:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
MAX_LEN = 125

train_ids = []
for sent in train_sents:
    subwords = '<s> ' + bpe.encode(sent) + ' </s>'
    encoded_sent = vocab.encode_line(subwords, append_eos=True, add_if_not_exist=False).long().tolist()
    train_ids.append(encoded_sent)

val_ids = []
for sent in val_sents:
    subwords = '<s> ' + bpe.encode(sent) + ' </s>'
    encoded_sent = vocab.encode_line(subwords, append_eos=True, add_if_not_exist=False).long().tolist()
    val_ids.append(encoded_sent)

train_ids = pad_sequences(train_ids, maxlen=MAX_LEN, dtype="long", value=0, truncating="post", padding="post")
val_ids = pad_sequences(val_ids, maxlen=MAX_LEN, dtype="long", value=0, truncating="post", padding="post")
```

```
train_ids
```

```
array([[    0, 30434,   957, ...,     0,     0,     0],
       [    0, 43062, 25556, ...,     0,     0,     0],
       [    0, 13072,   167, ...,     0,     0,     0],
       ...,
       [    0, 29340,   957, ...,     0,     0,     0],
       [    0, 34664, 54922, ...,     0,     0,     0],
       [    0, 13201,   718, ...,     0,     0,     0]])
```

Như vậy train_ids bây giờ đã trở thành một list dữ liệu mẫu trong đó mỗi mẫu là một list id của các subword có trong từ điển. Các câu ngắn hơn 125 subword được padding 0 ở cuối, những câu dài hơn được cắt đi cho đủ 125.

Tiếp theo, em tạo một mask gồm các giá trị 0, 1 để làm đầu vào cho thư viện transformers, mask này cho biết các giá trị nào của chuỗi đã được padding.

```
train_masks = []
for sent in train_ids:
    mask = [int(token_id > 0) for token_id in sent]
    train_masks.append(mask)

val_masks = []
for sent in val_ids:
    mask = [int(token_id > 0) for token_id in sent]

    val_masks.append(mask)
```

Sau khi chuẩn bị xong dữ liệu đầu và cho mô hình, bước cuối cùng của quá trình tiền xử lý dữ liệu sẽ là chuyển dữ liệu về tensor và sử dụng DataLoader của torch để tạo dataloader

4.2 Cơ sở lý thuyết

4.2.1 Naïve Bayes

4.2.1.1 Định lý Bayes

$$p(h|D) = \frac{P(D|h) \cdot P(h)}{P(D)}$$

Trong đó:

- P(h): Xác suất trước (tiên nghiệm) của giả thiết (phân loại) h.
- P(D): Xác suất trước (tiên nghiệm) của việc quan sát được dữ liệu D.
- P(D|h): Xác suất (có điều kiện) của việc quan sát được dữ liệu D, nếu biết giả thiết (phân loại) h là đúng.
- P(h|D): Xác suất (có điều kiện) của giả thiết (phân loại) h là đúng, nếu quan sát được dữ liệu D.

4.2.1.2 Phân loại Naïve Bayes:

Xét bài toán classification với C classes 1,2,…C. Giả sử có một điểm dữ liệu $x \in R^d$. Hãy tính xác suất để điểm dữ liệu này rơi vào class c.

$$P(y=c|x) = P(c|x) \quad (1)$$

Xác định class của điểm dữ liệu đó bằng cách chọn ra class có xác suất cao nhất:

$$c = \arg \max_{c \in \{1,\ldots,C\}} p(c|\mathbf{x}) \quad (2)$$

Tuy nhiên biểu thức trên thường khó tìm và khó thực hiện nên định lý Bayes được sử dụng

$$c = \arg\max_c p(c|\mathbf{x}) \ (3) \quad = \arg\max_c \frac{p(\mathbf{x}|c)p(c)}{p(\mathbf{x})} \quad 4) \quad = \arg\max_c p(\mathbf{x}|c)p(c) \ (5)$$

Từ (4) sang (5) bởi vì x không phụ thuộc vào c.

$$P(c) = \frac{\text{Điểm dữ liêu trainning vào class}}{\text{Tổng dữ liệu}} (MLE)$$

P(x|c) là phân phối điểm dữ liệu của c nên khó tính toán . Cho nên ta thường giả sử thành phần biến x độc lập với nhau

Cách giả sử này là Naïve Bayes. Cách xác định class dựa trên giả thiết này là Naïve Bayes Classifier.

Giải thuật:

- Tại bước training, dựa vào training data tính p(c), p(x_i|c) với i = 1, …, d
- Tại bước test, khi có một điểm dữ liệu x mới thì class

$$c = \arg \max_{c \in \{1,\ldots,C\}} p(c) \prod_{i=1}^{d} p(x_i|c)$$

- Khi d lớn và các xác suất nhỏ thì vế phải của biểu thức nhỏ và dễ có sai số nên ta thay bằng

$$c = \arg \max_{c \in \{1,\ldots,C\}} = \log(p(c)) + \sum_{i=1}^{d} \log(p(x_i|c))$$

Giả thiết trên có vẻ phi thực tết nhưng lại hoạt động rất tốt, đặc biệt là những bài toán phân loại văn bản.

4.2.1.3 Các phân phối thường dùng cho P(x_i|c)

Gaussian Naïve Bayes:

- Mô hình này được dùng chủ yếu cho dữ liệu mà các thành phần liên tục.Với mỗi chiều dữ liệu i và một class c, x_i tuân theo một phân phối chuẩn có kỳvọng μ_{ci} và phương sai σ_{ci}^2:

$$p(x_i|c) = p(x_i|\mu_{ci}, \sigma_{ci}^2) = \frac{1}{\sqrt{2\pi\sigma_{ci}^2}} \exp\left(-\frac{(x_i - \mu_{ci})^2}{2\sigma_{ci}^2}\right)$$

- Trong đó, bộ tham số $\theta = \{\mu_{ci}, \sigma^2\}$ được xác định bằng Maximum Likelihood:

$$(\mu_{ci}, \sigma_{ci}^2) = \arg \max_{\mu_{ci}, \sigma_{ci}^2} \prod_{n=1}^{N} p(x_i^{(n)}|\mu_{ci}, \sigma_{ci}^2)$$

Multinomial Naïve Bayes:

- Mô hình này chủ yếu được sử dụng trong phân loại văn bản mà feature vectorsđược tính bằng Bags of Words. Mỗi văn bản được biểu diễn bởi một vector có độ dài d chính là số từ trong từ điển. Giá trị của thành phần thứ i trong mỗi vector chính là số lần từ thứ i xuất hiện trong văn bản đó. Khi đó, p(xi|c) tỉ lệ với tần suất từ thứ i (hay feature thứ i cho trường hợp tổng quát)xuất hiện trong các văn bản của class c.

$$\lambda_{ci} = p(x_i|c) = \frac{N_{ci}}{N_c}$$

- Trong đó:
  - Nci là tổng số lần từ thứ i xuất hiện trong các văn bản của class c, nó được tính là tổng của tất cả các thành phần thứ I của các feature vectors tương ứng với class c.
  - Nc là tổng số từ (kể cả lặp) xuất hiện trong class c. Nói cách khác, nó bằng tổng độ dài của toàn bộ văn bản phụ thuộc vào class c.
  - Tuy nhiên cách tính này có một hạn chế là nếu có một từ chưa từng xuất hiện ở class c thì vế phải của biểu thức sẽ bằng 0 bất kể các giá trị còn lại lớn thế nào. Điều này sẽ dẫn đến kết quả không chính xác. Để giải quyết vấn đề này, một kỹ thuật được gọi là Laplace smoothing được áp dụng:

$$\hat{\lambda}_{ci} = \frac{N_{ci} + \alpha}{N_c + d\alpha}$$

Bernouli Naïve Bayes:

- Mô hình áp dụng cho dữ liệu mà mỗi thành phần là 1 giá trị binary $x_i \in \{0,1\}$

$$P(x_i|c) = P(i|c)^{xi} (1-P(i|c))^{1-xi}$$

4.2.2 Logistic Regression

4.2.2.1 Định nghĩa

Hồi quy logistic là một phương pháp phân tích thống kê được sử dụng để dự đoán giá trị dữ liệu dựa trên các quan sát trước đó của tập dữ liệu.

Mục đích của hồi quy logistic là ước tính xác suất của các sự kiện, bao gồm xác định mối quan hệ giữa các tính năng từ đó đự đoán xác suất của các kết quả, nên đối với hồi quy logistic ta sẽ có:

- **Input**: dữ liệu input (ta sẽ coi có hai nhãn là 0 và 1).
- **Output** : Xác suất dữ liệu input rơi vào nhãn 0 hoặc nhãn 1.

4.2.2.2 Xây dựng mô hình

a, Các biến và công thức trong mô hình

Xét trên một điểm dữ liệu ta có input x=[x1;x2;…;xn] sẽ là một vector cột, ta sẽ ngăn cách các feature(xi) bằng dấu ";". input x sẽ có dạng như bên dưới:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Xét trên toàn bộ bộ dữ liệu (m điểm dữ liệu) ta sẽ có một vector hàng $X=[x^{(1)},x^{(2)},\ldots,x^{(m)}]$ với mỗi cột là một điểm dữ liệu x^i ngăn cách nhau bởi dấu ",":

$$\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}] = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \ldots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \ldots & x_2^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \ldots & x_n^{(m)} \end{bmatrix}$$

Bộ trọng số w=[w_1;w_2;…;w_n] cũng sẽ là một vector cột, ta sẽ ngăn cách các trọng số w_i bằng dấu ";" bộ trọng số w sẽ có dạng như dưới:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Xét trên một điểm dữ liệu ta có:

$$z = \mathbf{w}^T \mathbf{x} + b$$
$$a = \sigma(z)$$

Xét trên toàn bộ tập dữ liệu ta đặt Z = [$z^{(1)}$, $z^{(2)}$,…,$z^{(m)}$] và A = [$a^{(1)}$, $a^{(2)}$,…, $a^{(m)}$], đây đều là các vector hàng với m là số điểm dữ liệu:

$$\mathbf{Z} = [z^{(1)}, z^{(2)}, \ldots, z^{(m)}]$$
$$\mathbf{A} = [a^{(1)}, a^{(2)}, \ldots, a^{(m)}]$$

Hàm kích hoạt là hàm sigmoid có công thức như sau:$\sigma(x) = \frac{1}{1+e^{-x}}$

b, Mô hình

Tất cả các dạng hồi quy thực tế có cùng chung một phương trình tổng quát để giúp chỉ ra rằng biến mục tiêu y(biến phụ thuộc) sẽ thay đổi như nào dựa vào mối quan hệ của nó với tất cả các biến độc lập x mà không quan tâm đến việc mối quan hệ đó là phi tuyến hay tuyến tính:

$$y = w_n x_n + w_{n-1} x_{n-1} + \cdots + w_1 x_1 + w_0$$

Trong đó:

- w0 là giá trị ước lượng của y khi tất cả các xi đều đạt giá trị 0.
- ϵ là sai số thể hiện các yếu tố chưa thể nghiên cứu đến nhưng các yếu tố này vẫn ảnh hưởng tới y.

- wi là tham số ước lượng(hay ta còn gọi là trọng số), có thể hiểu nôm na là mỗi wi sẽ chịu trách nhiệm quản lí một xi và wi sẽ điều chỉnh để có giá trị y mong muốn, lưu ý là giá trị tham số wi có thể thay đổi còn giá trị xi là cố định. Khi thiết lập mô hình hồi quy ta quan tâm đến quan hệ giữa nhiều biến độc lập xi với biến mục tiêu(biến phụ thuộc), phương trình tổng quát để ước lượng biến mục tiêu y:

$$E(y) = w_n x_n + w_{n-1} x_{n-1} + \cdots + w_1 x_1 + w_0 = \mathbf{w}^T \mathbf{x} + w_0$$

E(y) trong phương trình logistic regression là xác suất để kết luận giá trị của biến y không phải giá trị thực của biến y:

$$E(y) = P(y = 0||1|x_1, x_2, \ldots, x_n)$$

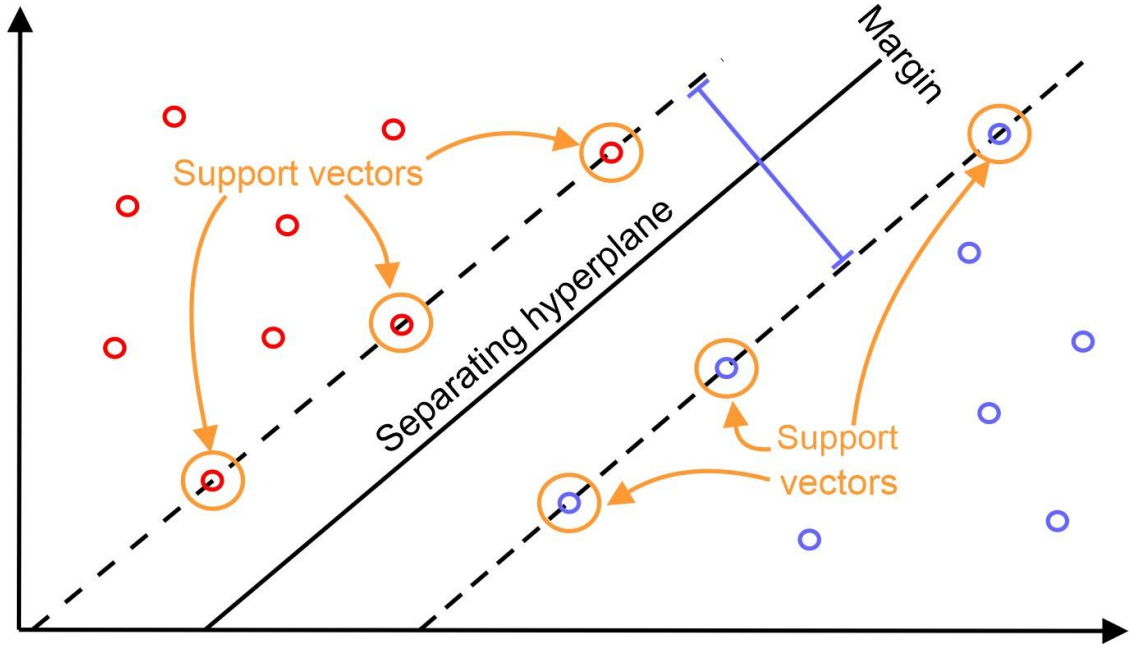
Áp dụng hàm sigmoid để chuyển giá trị w^Tx+w_0 thành xác suất để kết luận giá trị của biến y từ đó để xác định được nhãn của input x:

$$P = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x} + w_0)}}$$

Đây chính là mô hình của logistic regression.

4.2.3 Support Vector Machine

Support Vector Machine (Vapnik, 1995) viết tắt là SVM hay còn gọi là máy vector hỗ trợ là thuật toán thứ 3 được nhóm sử dụng cho việc phân loại bài báo. Trong mục này em sẽ không đi quá kĩ vào việc phân tích SVM tuy nhiên vẫn sẽ tóm lược lại tư tưởng chính của thuật toán.



Hình 3. Minh họa thuật toán SVM

Về cơ bản, SVM đi tìm một siêu phẳng tuyến tính phân tách giữa các điểm dữ liệu thuộccác lớp khác nhau. Siêu phẳng này có điểm đặc biệt so với những mặt phân cách sử dụngcác thuật toán khác. Gọi những điểm dữ liệu gần với mặt phân cách nhất là các vector hỗtrợ, các vector hỗ trợ này nằm trên một mặt biên song song với mặt phẳng phân cách và khoảng cách giữa chúng với mặt phân cách được gọi là lề. SVM luôn đi tìm cho ta mặt phân cách có lề lớn nhất. Mặt phân cách này cũng đặc biệt được xây dựng chỉ bởi các vector hỗ trợ. Điều này có được trong quá tình tối ưu giá trị lề và sử dụng các điều kiện Karush-Kuhn-Tucker.

$$f(x) = \beta_0 + \beta^T x = 0$$

Công thức của mặt phẳng phân cách của SVM

$$\min_{\beta,\beta_0} \frac{1}{2}||\beta||^2$$

$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1,\ i = 1, \ldots, N.$$

Bài toán tối ưu đi tìm tham số cho mặt phẳng phân cách của SVM. Trong việc tối ưu, cóthể thấy giá trị beta được xây dựng bởi các vector hỗ trợ.

Tuy nhiên nếu chỉ dừng lại ở trên SVM sẽ chỉ dừng lại ở việc phân loại được các bộ dữ liệuthực sự tách biệt tuyến tính. Với những bộ dữ liệu không tách biệt tuyến tính, SVM đưa ra giải pháp là cho phép mỗi điểm dữ liệu có thể đi quá mặt biên, tuy nhiên sẽ giới hạn việc điquá mặt biên lại bằng việc lấy cận trên của tổng khoảng cách đi qua mặt biên là một giá trị K, đại diện cho việc K điểm đi quá mặt phẳng phân cách.

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N} \xi_i$$

$$\text{subject to} \quad \xi_i \geq 0,\ y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i\ \forall i,$$

Epsilon i là đại lượng biểu diễn tỷ lệ đi quá biên của điểm dữ liệu thứ i. Cụ thể là tỷ lệ giữa khoảng cách đi quá và lề, như vậy nếu giá trị nó lớn hơn 1 tức là nó đi quá mặt biên một đoạn lề hay chính là đi quá mặt phẳng phân cách. C là một hyperparameter thể hiện mức độchịu lỗi của thuật toán, C lớn mang nghĩa là cho phép ít điểm sai qua biên, C nhỏ tức là chophép nhiều điểm đi qua biên, C → vô cùng là trường hợp dành cho dữ liệu tách biệt tuyến tính.

Đối với các bộ dữ liệu không phân biệt tuyến tính, ngoài cho phép chịu lỗi như trên, SVM có đề xuất chiếu các điểm trong không gian ban đầu qua một không gian khác thông qua các hàm nhân. Không gian mới này đảm bảo tính tách biệt tuyến tính của dữ liệu, như vậy SVM có thể hoạt động hiệu quả. Điểm yếu của hướng đi này là cần tìm ra hàm nhân phù hợp với các thuộc tính của bộ dữ liệu. Việc tìm kiếm hàm nhân thì khônghề đơn giản, do đó nhóm chọn linear SVM hay SVM với nhân linear làm giải thuật cho việc phân loại. Linear SVM trong bài toán phân loại bài báo thể hiện rất tốt do dữ liệu văn bản sau khi nhúng thể hiện tích tách biệt tuyến tính một cách tương đối, đồng thời SVM cũng cho tốc độ tính toán rất cao do chỉ xây dựng mặt phẳng phân loại dựa trên một phần nhỏ điểm dữ liệu, chính là các vector hỗ trợ. Ngoài Linear SVM nhóm quyết định xây dựng một mô hình SVM phi tuyến với nhân radial basis function để thí nghiệm xem, kernel SVM này có tốt hơn so với linear SVM hay không.

$$\text{radial basis function (RBF): } K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2),\ \gamma > 0$$

4.2.4 Long Short Term Memory

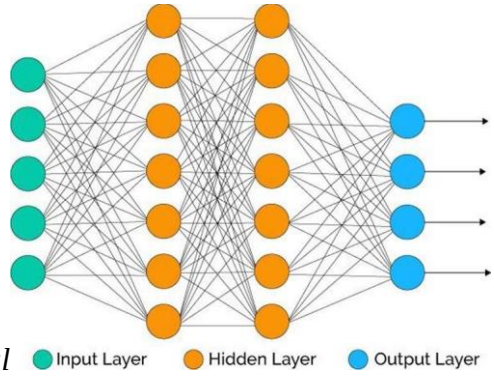
4.2.4.1 Mạng Neural

Neural Network đọc tiếng việt là Mạng nơ-ron nhân tạo, đây là một chuỗi những thuật toán được đưa ra để tìm kiếm các mối quan hệ cơ bản trong tập hợp các dữ liệu. Thông qua việc bắt bước cách thức hoạt động từ não bộ con người.

Mạng nơ ron nhân tạo có thể hoạt động như mạng nơ ron của con người. Mỗi một nơ ron thần kinh trong nơ ron nhân tạo là hàm toán học với chức năng thu thập và phân loại các thông tin dựa theo cấu trúc cụ thể.

Mạng Neural là sự kết hợp của những tần perceptron hay còn gọi là perceptron đa tầng. Và mỗi một mạng Neural sẽ thường bao gồm ba kiểu tầng là:

- Tầng input layer (tầng vào): Tầng này nằm ở bên trái cùng của mạng, thể hiện cho các đầu vào của mạng.
- Tầng output layer (tầng ra): Là tầng nằm bên phải cùng của mạng, thể hiện cho đầu ra của mạng.
- Tầng hidden layer (tầng ẩn): Tầng này nằm giữa tầng vào và tầng ra của mạng, thể hiện cho quá trình suy luận logic của mạng.
- Mỗi một Neural Network chỉ bao gồm 1 tầng vào (input layer) và 1 tầng ra (output layer) nhưng có thể có rất nhiều tầng ẩn (hidden layer):

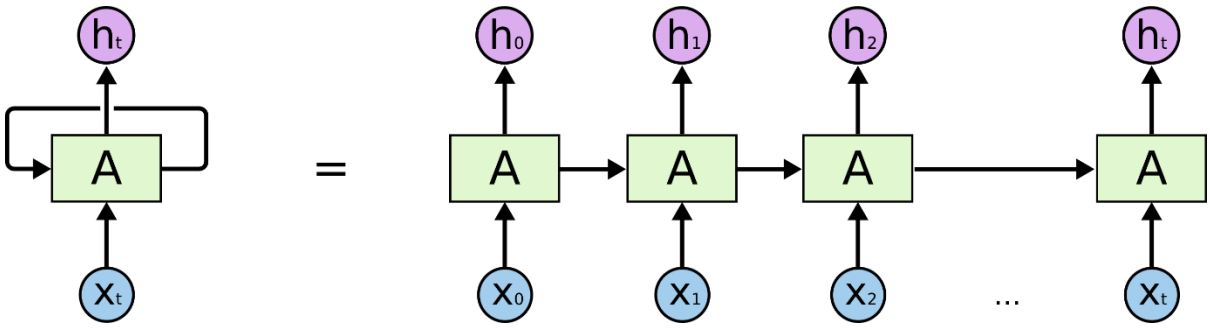


Hình 4. Kiến trúc mạng Neural

4.2.4.2 Mạng hồi quy

Trong thực tế có một vấn đề mà mạng nơ-ron truyền thống gặp phải chính là khả năng lưu trữ lại những gì diễn ra trước đó và điều này dẫn đến bất lợi trong khá nhiều bài toán trong lĩnh vực trí tuệ nhận tạo nói chung và xử lý ngôn ngữ tự nhiên nói

riêng. Vì thế mạng nơ-ron hồi quy (Recurrent Neural Network) sinh ra để giải quyết vấn đề đó. Mạng này chứa các vòng lặp bên trong cho phép thông tin có thể lưu lại được. Các vòng lặp này khiến cho mạng nơ-ron hồi quy trông có vẻ khó hiểu. Tuy nhiên, nếu thực tế thì nó không khác nhiều so với các mạng nơ-ron thuần túy. Một mạng nơ-ron hồi quy có thể được coi là nhiều bản sao chép của cùng một mạng, trong đó mỗi đầu ra của mạng này là đầu vào của một mạng sao chép khác. Hình mô tả sau sẽ giúp chúng ta phần nào hiểu được vấn đề:



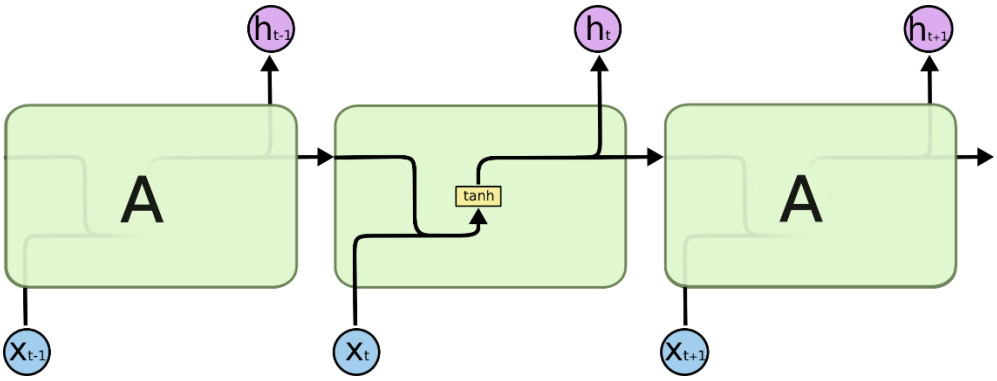
Hình 5. Kiến trúc mạng RNN

4.2.4.3 Mạng LSTM

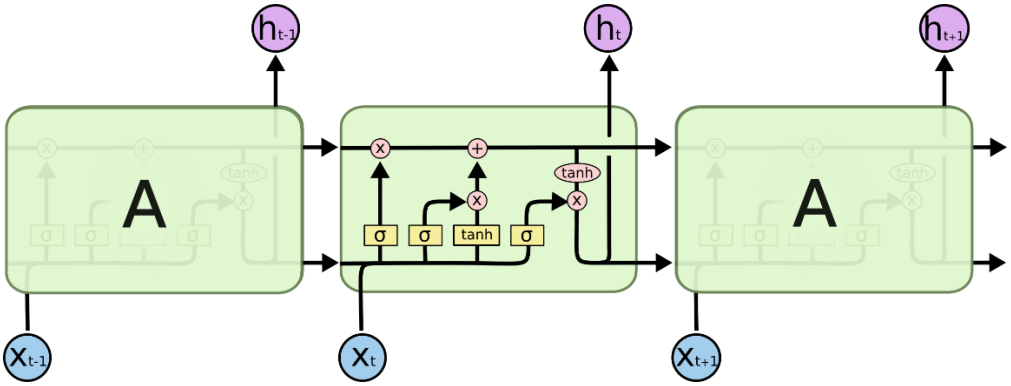
RNN được ứng dụng và thành công ở rất nhiều bài toán, đặc biệt là ở lĩnh vực NLP(xử lý ngôn ngữ tự nhiên). Trên lý thuyết thì đúng là RNN có khả năng nhớ được những tính toán (thông tin) ở trước nó, nhưng mô hình RNN truyền thống không thể nhớ được những bước ở xa do bị mất mát đạo hàm (vanishing gradient) nên do đó mô hình cả tiến có thể xử lý được vấn đề phụ thuộc xa là LSTM ra đời. LSTM về cơ bản cũng giống với RNN truyền thống ngoài việc thêm các cổng tính toán ở hidden layer để quyết định giữ lại các thông tin nào.

Trong bài toán này ngoài các thuật toán học máy cơ bản được em sử dụng thì LSTM sẽ là thuật toán học sâu tiếp theo được dùng trong quá trình huấn luyện và kiểm thử cho bài toán để có thể đánh giá chất lượng từng mô hình.

2 hình ảnh dưới đây sẽ cho thấy được sự khác nhau tương quan giữa LTSM và RNN với hình ảnh đầu tiên là một mạng RNN và hình ảnh thứ 2 là một mạng LSTM.



Hình 6. Mạng RNN



Hình 7. Mạng LSTM

4.2.5 Transformer

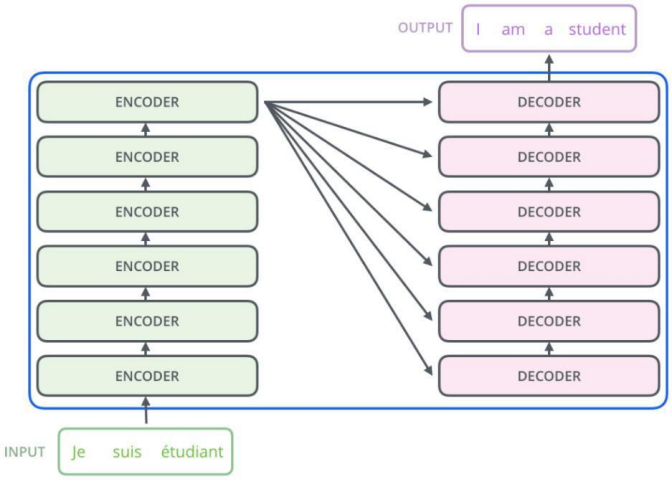
4.2.5.1 Tổng quan về Transformer

Transformer là một mô hình học sâu được giới thiệu vào 2017, rất hay được sử dụng cho bài toán tóm tắt văn bản theo hướng tiếp cận abstraction, làm việc với dữ liệu dạng sequence (seq). Đây là một kiến trúc khá trừu tượng với phương châm "Attention is all we need"

Trước khi Transformer ra đời, hướng tiếp cận truyền thống đối với các bài toán seq2seq là dùng các kiến trúc mạng RNN, LTSM, GRU… Khi đó, người ta dùng RNNs cho cả Encoder và Decoder.

Học sâu ngày càng phát triển, lợi thế tính toán song song của GPU cũng cần đượctận dụng tối đa để tăng tốc độ train, ngoài ra còn cần phải giải quyết vấn đề vanishing- gradient khi gặp phải các câu dài. Và Transformer đã giải quyết được những vấn đề đó.

4.2.5.2 Hai thành phần của Transformer



Hình 8. Các tầng của Encoders và Decoders xét theo chiều đi từ dưới lên

Transformer được cấu thành bởi 2 thành phần là Encoders và Decoders:

a, Encoders:

- Chức năng: thực hiện chuyển từng từ trong câu gốc thành các vector để biểu diễn input.
- Mô tả: Encoders bao gồm các encoder chồng lên nhau tạo thành 1 stack, hướng từ dưới lên (như hình trên). Trong đó, encoder ở dưới cùng nhận đầu vào dưới dạng các word embedding, các encoder bên trên còn lại nhận đầu vào là output của encoder bên dưới nó. Các encoder khác tầng có cấu trúc giống nhau nhưng trọng số khác nhau.
- Cấu trúc và hoạt động: Ở mỗi tầng encoder gồm nhiều layer, mỗi layer xử lý input và sinh ra output cho layer bên trên. Đầu tiên, input của một encoder đi qua self- attention layer. Nhiệm vụ của self-attention layer là tính các output (mỗi output tương ứng với 1 input) bằng tổng trọng số của tất cả các input (cơ chế self-attention). Output của self-attention layer sẽ được chuyển lên cho feed forward neural network layer.

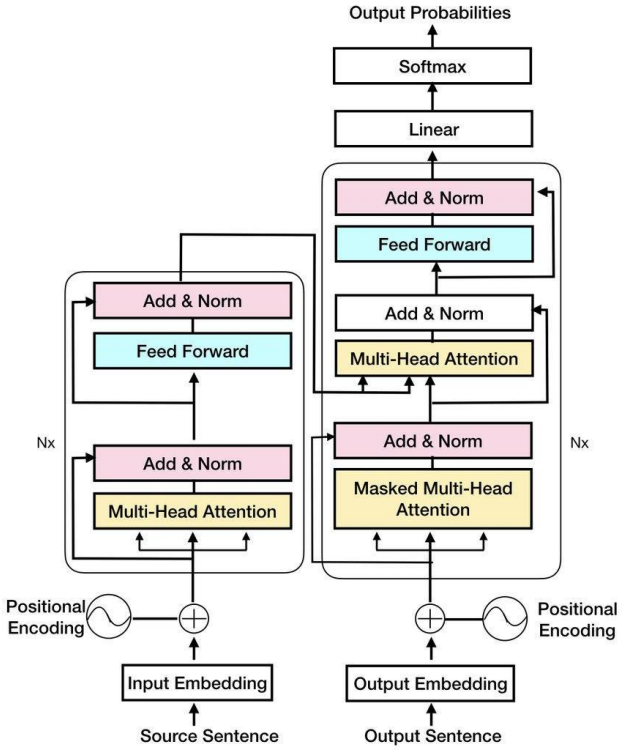
b, Decoders:

- Chức năng: thực hiện chuyển các vector mà Encoders gửi cho thành output là câu đích hoàn chỉnh.
- Mô tả: Decoders bao gồm các decoder chồng lên nhau tạo thành 1 queue. Trong đó,Input của tầng decoder dưới cùng là các word embedding và output của Encoders; input của các tầng decoder còn lại là output của

decoder nằm dưới nó và output của Encoders. Các decoder khác tầng có cấu trúc giống nhau nhưng trọng số khác nhau.

- Cấu trúc và hoạt động: Một decoder có cấu trúc gần giống với encoder. Điểm khác biệt ở đây là decoder là có thêm 1 Encoder-Decoder self-attention layer ở giữa self-attention layer và feed forward neural network layer, cộng với layer dưới cùng sử dụng masked multi-head attention thay vì multi-head attention (che đi các từ ở tương lai chưa được mô hình dịch đến- nhân với một vector chứa các giá trị 0,1). Encoder- Decoder self-attention layer sử dụng các giá trị query từ self-attention layer, các giá trị key và value của input được tính từ output của Encoders, có nhiệm vụ so sánh sự tương quan giữa các từ đang được dịch với các từ nguồn; việc tính toán sử dụng multi-heads attention.
- Sau đó: Output của Decoders sẽ được đưa vào một tầng neural network để tính vectorsoftmax rồi đưa ra xác suất của từ tiếp theo trong câu đích.
- Tất cả các từ đã được sinh ra (word embedding + positional encoding cho mỗi từ) sẽđược cho lại vào Encoders để sinh ra từ tiếp theo. Quá trình này lặp lại và dừng khi xuất hiện 1 token đặc biệt đánh dấu rằng câu kết thúc.

4.2.5.3 Các cơ chế hoạt động của Transformer



Hình 9: Kiến trúc chi tiết của Transformer

a, Embedding layer with Positional Encoding

Trong Transformer, input được đẩy vào cùng 1 lúc và không còn khái niệm time-step như trong RNNs. Các câu được đẩy vào sẽ cần được chuyển về dạng vector/ ma trận, gọi là input embedding (có thể dùng GloVe, Word2Vec…).

Do vị trí của từ trong câu sẽ có ảnh hưởng đến ngữ nghĩa, mặt khác input lại đượcđẩy song song cùng 1 lúc, vì thế cần có cơ chế để lưu lại - ở đây là vector Positional Embedding (P E) với độ dài bằng độ dài các vector input embedding.

Các vector được đẩy vào sẽ là các vector tổng của các vector input embedding với các vector PE, biểu diễn được ngữ nghĩa của từ và nghĩa cảnh của nó trong câu. Côngthức của PE:

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{\text{model}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{\text{model}}}\right)$$

(pos là vị trí của từ trong câu PE là giá trị phần tử thứ i trong embeddings có độ dài dmodel)

b, Self-attention, Multi-head-attention, Masker-Multi-head-attention

Được xem như là linh hồn của Transformer. Nó là một phiên bản của soft-attention (họcbộ trọng số bằng thuật toán backpropagation), giúp tạo ra quan hệ giữa các từ trong câu theo ngữ cảnh cụ thể.

Theo hình 2, có 3 mũi tên từ input embedding hướng đến multi-head-attention

$$Z = \text{softmax}\left(\frac{Q \cdot K^1}{\sqrt{Dimension\ of\ vector}\ Q, K \text{ or } V}\right) \cdot V$$

layer, thểhiện 3 vector Queries (Q), Keys (K), Values (V), từ đó ta tính được vector attention :

Trong đó : Q, K, V bằng input embeddings nhân với ma trận trọng số tương ứng (được tune trong quá trình train) WQ, WK, WV. Bây giờ, K đóng vai trò như khóa đại diện chotừ, Q cho phép truy vấn đến các K của các từ trong câu bằng cách nhân chập với các K. Mục đích của phép nhân chập là để tính độ liên quan giữa các từ với nhau. Hai từ liên quan đến nhau nhiều hơn sẽ có *Score*=$Q.K^T$ lớn hơn. Việc chia cho Dimension như trên giúp *Score* không phụ thuộc độ dài

vector Q/K/V. Sau đó là tính Softmax để thu được phân bố xác suất trên các từ. tiếp theo ta nhân phân bố xác suất đó với vector V để loại bỏ những từ không cần thiết (xác suất nhỏ) và giữ lại những từ quan trọng. Cuối cùng, các vectors V (đã được nhân với softmax output) cộng lại với nhau, tạo ra vector attention Z cho một từ. Lặp lại quá trình trên cho tất cả các từ ta được ma trận attention cho 1 câu.

Về self-attention, một từ sẽ luôn chú ý vào chính nó, vì nó liên quan đến chính nó nhiều nhất. Nhưng điều chúng ta cần là sự tương tác giữa các từ khác nhau trong câu, multi- head-attention – phiên bản nâng cấp của self-attention sẽ giải quyết vấn đề này. Cách hoạt động của multi-head là thay vì dùng 1 self-attention (1 head) thì ta dùng 8 head. Saudó để tạo ra 1 ma trận attention duy nhất như self-attention, ta cần concat 8 ma trận với nhau rồi nhân với ma trận trọng số WO (được tune trong khi training).

c, The Residuals:

Mỗi sub-layer đều là một residual block, cho phép skip connections (cho thông tin đi qua sub-layer trực tiếp). Thông tin này (x) được cộng với attention (z) của nó và thực hiện Layer Normalization. Trong kiến trúc của mô hình Transformer, residuals connection và normalization layer được sử dụng mọi nơi. Hai kỹ thuật này giúp cho môhình huấn luyện nhanh hội tụ hơn và tránh mất mát thông tin

d, Feed forward neural network

Sau khi được Normalize, các vectors z được đưa qua mạng fully connected này trướckhi đẩy qua Decoder. Vì các vectors này không phụ thuộc vào nhau nên ta có thể tậndụng được tính toán song song cho cả câu

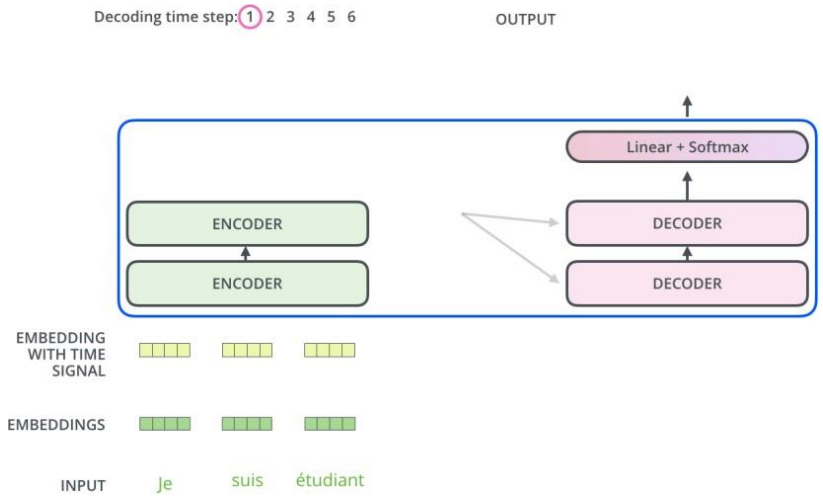
Gồm 2 linear transform và 1 ReLu activation ở giữa: FFN(x) = max(0,xW1 + b1)W2 + b2

Các tham số ma trận neural network của các tầng là khác nhau

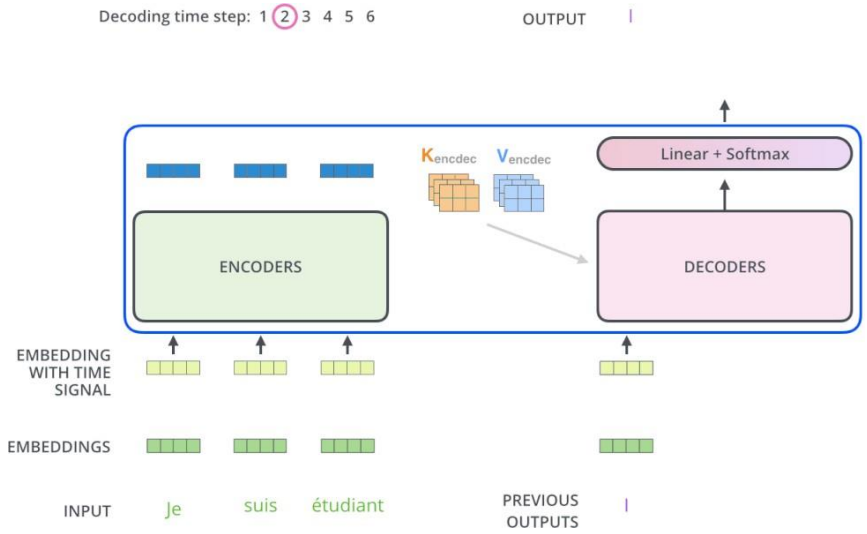
e, Quá trình decode

Quá trình decode về cơ bản là giống với encode, chỉ khác là Decoder decode từng từ mộtvà input của Decoder bị masked. Sau khi masked input đưa qua sub-layer #1 của Decoder, nó sẽ không nhân với 3 ma trận trọng số để tạo ra Q, K, V nữa mà chỉ nhân với1 ma trận trọng số WQ. K và V được lấy từ Encoder cùng với Q từ Masked multi-head attention đưa vào sub-layer #2 và #3 tương tự như

Encoder. Cuối cùng, các vector được đẩy vào lớp Linear (là mạng Fully Connected) theo sau bởi Softmax để cho ra xác suấtcủa từ tiếp theo.



Hình 10. Encoding



Hình 11. Decoding

4.2.6 BERT

4.2.6.1 Tổng quan về BERT

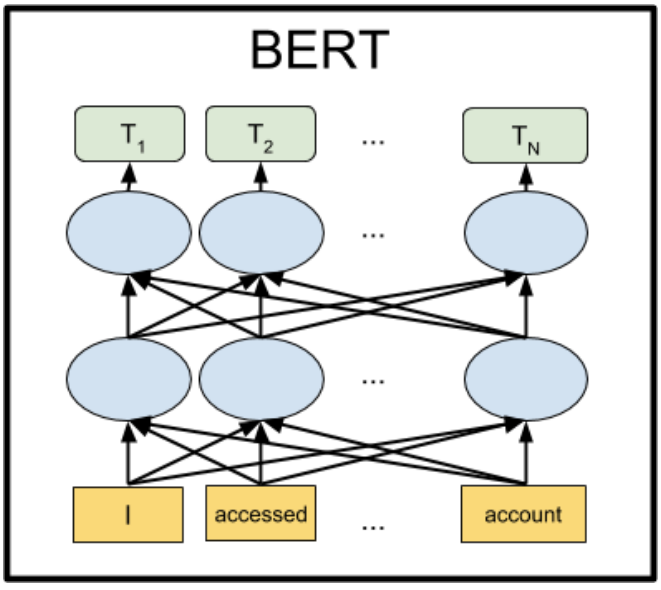
BERT (Bidirectional Encoder Representations from Transformers) cũng là một model biểu diễn ngôn ngữ trong NLP, được Google trình làng vào 2018.

Công cụ này tỏ ra vô cùng hiệu quả trong mảng NLP bởi những thành tích chưa từng đạt được như tăng độ chính xác trở nên khá cao so với những mô hình ban đầu trên dữ liệu đánh giá GLUE (General Language Understanding Evaluation).

Lí do khiến BERT vượt xa RNN là vì nó nhúng thêm ngữ cảnh vào trong các vector input embedding. Về các phương pháp embedding thì gồm không ngữ cảnh (Worrd2Vec), ngữ cảnh 1 chiều (RNN), ngữ cảnh chạy theo 2 chiều độc lập - cũng không khá hơn 1 chiều là mấy (LSTM), và ngữ cảnh 2 chiều (BERT). Chính vì vậy khi che 1 từ đi thì lập tức model có thể predict được với độ chính xác cao dựa vào cả câu.

4.2.6.2. Kiến trúc của BERT

BERT đơn giản chỉ là phần Encoders của Transformer, input là các word embedding và output là các encoder output. Về kiến trúc của Transformer đã trình bày rất cụ thể ở trên.



Hình 12. Kiến trúc của BERT

4.2.6.3 Trainning

BERT là Transformer chỉ còn lại Encoders, vậy train như thế nào ?

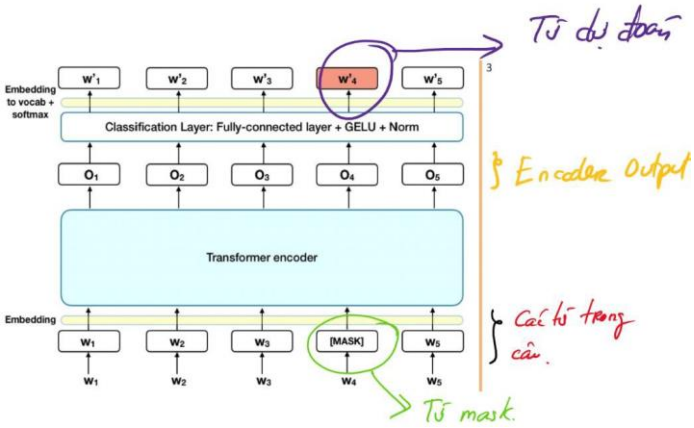
BERT được train đồng thời 2 task gọi là *Masked LM* (để dự đoán từ thiếu trong câu) và *Next Sentence Prediction* (NSP – dự đoán câu tiếp theo câu hiện tại). Hai task này được train đồng thời và loss tổng sẽ là kết hợp loss của 2 task và model sẽ cố gắng minimize loss tổng này. Chi tiết 2 task này như sau:

a, Với Masked LM:

Ta train sẽ thực hiện che đi tầm 15% số từ trong câu và đưa vào model. Và ta sẽ train để model predict ra các từ bị che đó dựa vào các từ còn lại.

Cụ thể là:

- Thêm một lớp classification lên trên encoder output
- Đưa các vector trong encoder ouput về vector bằng với vocab size, sau đó softmax đểchọn ra từ tương ứng tại mỗi vị trí trong câu.
- Loss sẽ được tính tại vị trí masked và bỏ qua các vị trí khác (để đánh giá xem modeldự đoán từ mask đúng/sai như thế nào, các từ khác là không liên quan).



Hình 13. Trainning

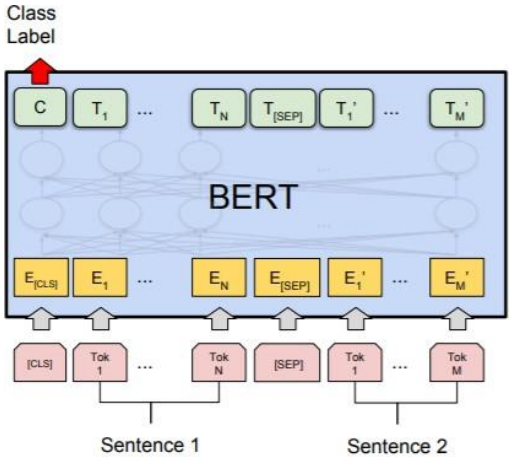
b, Với Next Sentence Prediction (NSP)

Model sẽ được feed cho một cặp câu và nhiệm vụ của nó là output ra giá trị 1 nếu câu thứ hai đúng là câu đi sau câu thứ nhất và 0 nếu không phải. Trong quá trinh train, ta chọn 50% mẫu là Positive (output là 1) và 50% còn lại là Negative được ghép ngẫu nhiên (output là 0).

Cụ thể cách train như sau:

- Bước 1: Ghép 2 câu vào nhau và thêm 1 số token đặc biệt để phân tách các câu. Token [CLS] thêm vào đầu cầu Thứ nhất, token [SEP] thêm vào cuối mỗi câu. Ví dụghép 2 câu "Hôm nay em đi học" và "Học ở trường rất vui" thì sẽ thành [CLS] Hômnay em đi học [SEP] Học ở trường rất vui [SEP]
- Bước 2. Mỗi token trong câu sẽ được cộng thêm một vector gọi là Sentence Embedding, thực ra là đánh dấu xem từ đó thuộc câu Thứ nhất hay câu thứ 2. Ví dụ:nếu thuộc câu Thứ nhất thì cộng thêm 1 vector toàn số "0" có kích thước bằng WordEmbedding, và nếu thuộc câu thứ 2 thì cộng thêm một vector toàn số "1".
- Bước 3. Các từ trong câu đã ghép sẽ được thêm vector Positional Encoding vào đểđánh dấu vị trí từng từ trong câu đã ghép (xem lại tại phần Transformer).
- Bước 4. Đưa chuỗi sau bước 3 vào mạng.
- Bước 5. Lấy encoder output tại vị trí token [CLS] được transform sang một vector có2 phần tử [c1 c2].
- Bước 6. Tính softmax trên vector đó và output ra probality của 2 class: Đi sau vàKhông đi sau. Để thể hiện câu thứ hai là đi sau câu thứ nhất hay không, ta lấy Argmax.

Kết quả thu được là một Pretrain model.



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

Hình 14. Cách lấy output đầu ra

4.2.7 Mô hình phoBERT

4.2.7.1 Tổng quan về RoBERTa

RoBERTa được giới thiệu bởi Facebook là một phiên bản được huấn luyện lại của BERT với một phương pháp huấn luyện tốt hơn với dữ liệu được tăng gấp 10 lần.

RoBERTa được xây dựng dựa trên chiến thuật mask ngôn ngữ của BERT, trong đó hệ thống sẽ học được cách dự đoán một số phần văn bản được chủ ý giấu đi trong số rất nhiều các văn bản không chú thích khác. Để tăng cường quá trình huấn luyện, RoBERTa không sử dụng cơ chế dự đoán câu kế tiếp (NSP) từ BERT mà sử dụng kỹ thuật mặt nạ động (dynamic masking), theo đó các token mặt nạ sẽ bị thay đổi trong quá trình huấn luyện. Sử dụng kích thước batch lớn hơn cho thấy hiệu quả tốt hơn khi huấn luyện. RoBERTa được thực hiện trên PyTorch, với khả năng thay đổi một số siêutham số chính trong BERT, trong đó bao gồm mục tiêu tiền huấn luyện câu tiếp theo của BERT, cũng như việc huấn luyện theo nhóm nhỏ và tốc độ học.

Một điều quan trọng nữa, RoBERTa sử dụng 160GB văn bản để huấn luyện. Trong đó,16GB là sách và Wikipedia tiếng Anh được sử dụng trong huấn luyện BERT. Phần cònlại bao gồm CommonCrawl News dataset (63 triệu bản tin, 76 GB), ngữ liệu văn bản Web (38 GB) và Common Crawl Stories (31 GB). Mô hình này được huấn luyện với GPU của Tesala 1024 V100 trong một ngày.

Kết quả là, RoBERTa vượt trội hơn BERT trên dữ liệu đánh giá GLUE (General Language Understanding Evaluation) :

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$BERT_{LARGE}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$XLNet_{LARGE}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-
Ensembles on test (from leaderboard as of July 25, 2019)										
ALICE	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	**96.8**	**93.0**	67.8	91.6	**90.4**	88.4
RoBERTa	**90.8/90.2**	**98.9**	90.2	**88.2**	96.7	92.3	67.8	**92.2**	89.0	**88.5**

Hình 15. Đánh giá Trích từ bài báo RoBERTa: A Robustly Optimized BERT Pretraining Approach.

4.2.7.2 phoBERT

Mô hình phoBERT là một pre-trained model được huấn luyện dành riêng cho tiếngViệt. Cách tiếp cận giống RoBERTa của Facebook được giới thiệu 2019.

Tương tự như BERT, PhoBERT cũng có 2 phiên bản là PhoBERTbase với 12 transformers block và PhoBERTlarge với 24 transformers block. PhoBERT được train trên khoảng 20GB dữ liệu bao gồm khoảng 1GB Vietnamese Wikipedia corpus và 19GB còn lại lấy từ Vietnamese news corpus. Đây là một lượng dữ liệu khả ổn để trainmột mô hình như BERT.

Như đã nói ở trên, do tiếp cận theo tư tưởng của RoBERTa, PhoBERT chỉ sử dụng task Masked Language Model để train, bỏ đi task Next Sentence Prediction.

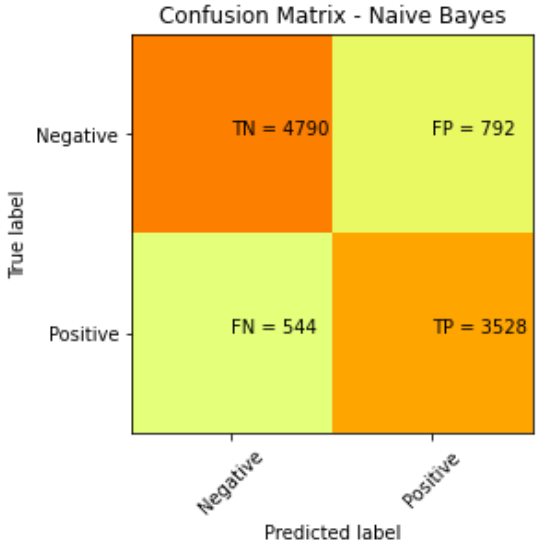
PhoBERT chính là mô hình được em sử dụng cho bài toán nhận dạng cảm xúc được quy về bài toán phân loại văn bản này.

5. Kết quả thực nghiệm

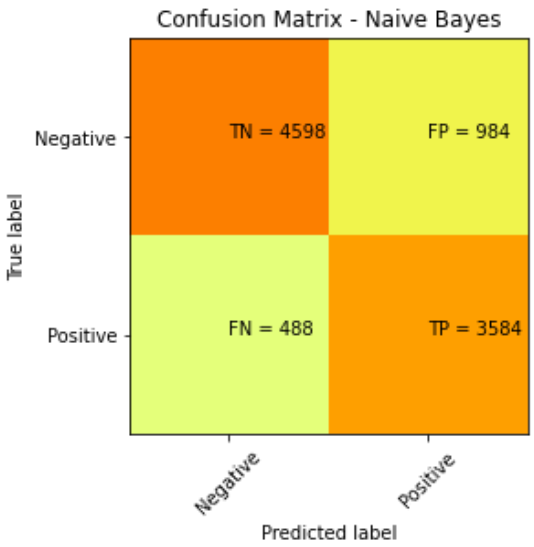
5.1 Naïve Bayes

Kết quả của mô hình Naïve Bayes được đánh giá trên tập kiểm thử với 2 kiểu mã hóa one-hot là count-based vector và TF-IDF vector sẽ có giá trị lần lượt là count_based accuracy = 0.85 và TF-IDF accuracy = 0.86.

Dưới đây là confusion matrix với kiểu mã hóa count-based vector và TF-IDF vector của Naïve Bayes:



Hình 16.Confusion Matrix Naïve Bayes(TF-IDF)

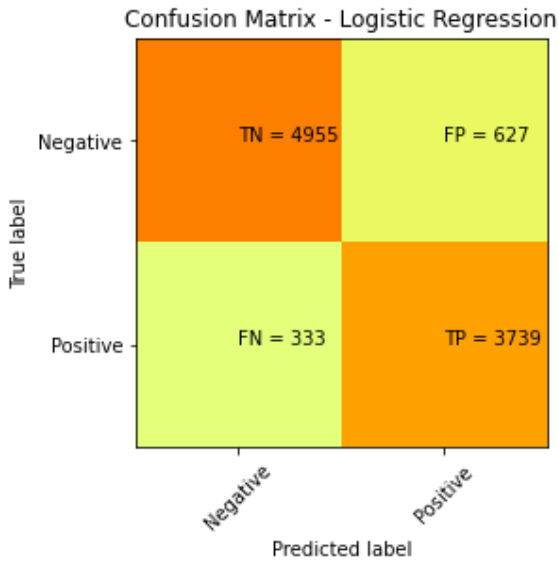


Hình 17.Confusion Maxtrix Naïve Bayes(CB)

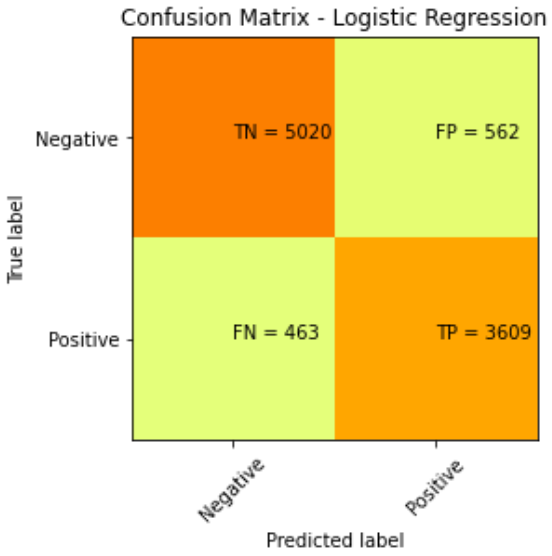
5.2 Logistic Regression

Kết quả của mô hình Logistic Regression được đánh giá trên tập kiểm thử với 2 kiểu mã hóa one-hot là count-based vector và TF-IDF vector sẽ có giá trị lần lượt là count_based accuracy = 0.89 và TF-IDF accuracy = 0.90

Dưới đây là confusion matrix với kiểu mã hóa count-based vector và TF-IDF vector của Naïve Bayes:



Hình 18.Confusion Matrix LR(TF-IDF)



Hình 19.Confusion Maxtrix LR (CB)

5.3 Support Vector Machine (SVM)

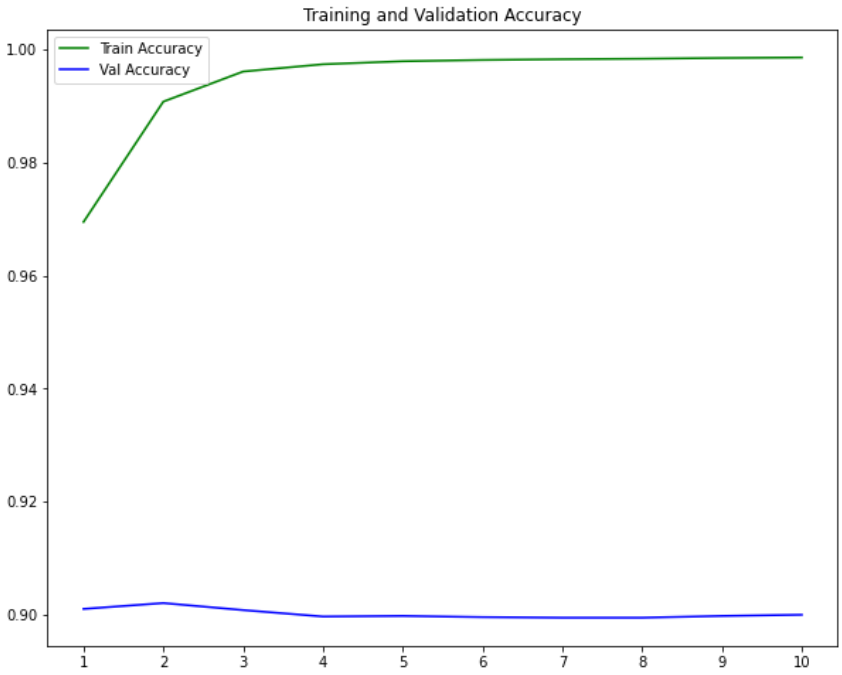
Đối với SVM nhân "RBF", em có thực hiện tuning một tham số đó là C. Nhắc lại, C là một hyperparameter thể hiện mức độ chịu lỗi của thuật toán, C lớn mang nghĩa là cho phép ít điểm sai qua biên, C nhỏ tức là cho phép nhiều điểm đi qua biên, C →vô cùng là trường hợp dành cho dữ liệu tách biệt tuyến tính.

Qua các quá trình thử nghiệm các mô hình Naïve Bayes và Logistic Regression với 2 kiểu mã hóa one-hot là count-based vector và TF-IDF vector thì kiểu mã hóa dữ liệu TF-IDF method đem lại kết quả tốt hơn trên cả 2 mô hình. Vì thế quá trình tuning tham số chỉ được thực hiện trên tập kiểm thử với kiểu mã hóa TF-IDF.

Ban đầu em thực hiện chấm điểm trên 4 giá trị C là 0.01, 0.1, 1.0, 10.0:

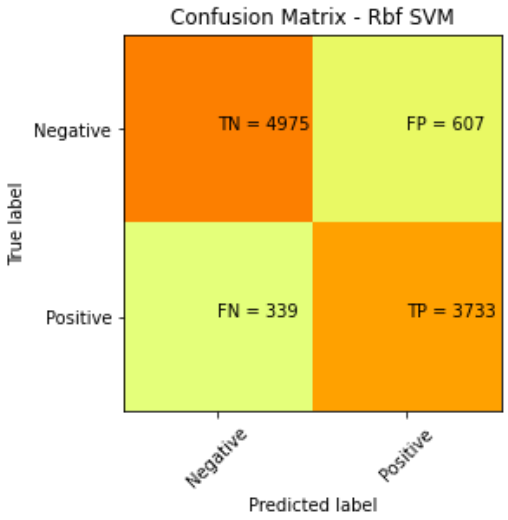
C	0.01	0.1	1.0	10.0
Train Accuracy	0.84	0.90	0.97	1.00
Val Accuracy	0.83	0.88	0.90	0.90

Nhận thấy kết quả val accuracy khá tốt trên 1 đến 10, em tiếp tục tuning trên khoảng này. Dưới đây là đồ thị thể hiện điểm số khi thực hiện tuning C từ 1 đến 10:



Hình 20. Tuning giá trị của C trong khoảng từ 1 đến 10

Sau khi chấm điểm, C = 2 cho kết quả cao nhất trên val accuracy. Cụ thể train accuracy = 0.9908 và val accuracy = 0.9020. Dưới đây là confusion matrix của "RBF" SVM với C = 2:

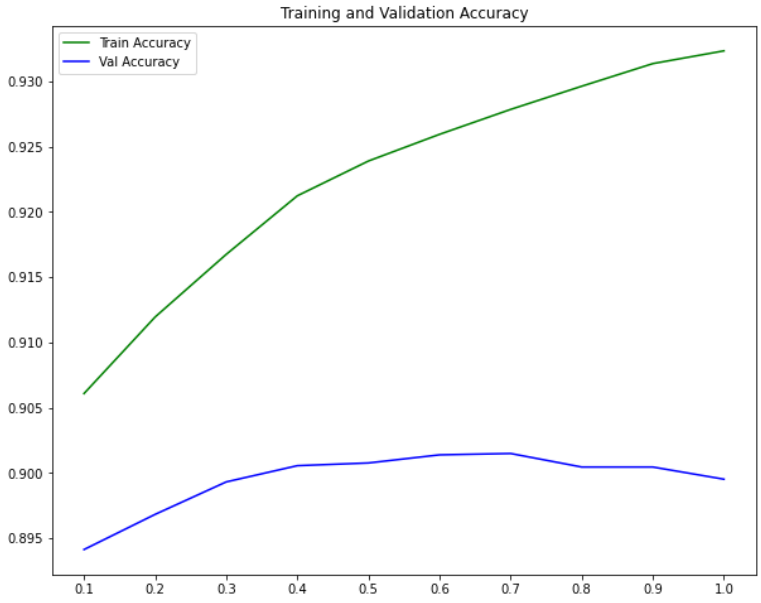


Hình 21. Confusion Matrix – "RBF" SVM (C=2)

Đối với SVM nhân "linear", em cũng thực hiện tuning tham số C. Đầu tiên em sẽ tiến hành chấm điểm trên 5 giá trị C là 0.01, 0.1, 1, 10, 100:

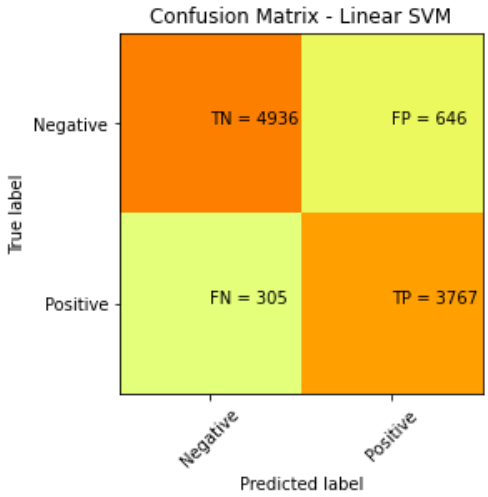
C	0.01	0.1	1.0	10.0	100.0
Train Accuracy	0.8695	0.9060	0.9323	0.9534	0.9618
Val Accuracy	0.8625	0.8941	0.8995	0.8864	0.8702

Nhận thấy kết quả val accuracy khá tốt trên 0.1 đến 1.0, em tiếp tục tuning trên khoảng này. Dưới đây là đồ thị thể hiện điểm số khi thực hiện tuning C từ 0.1 đến 1.0:



Hình 22. Tuning giá trị của C trong khoảng từ 0.1 đến 1.0

Sau khi chấm điểm, C = 0.7 cho kết quả cao nhất trên val accuracy. Cụ thể train accuracy = 0.9278 và val accuracy = 0.9015. Dưới đây là confusion matrix của "Linear" SVM với C = 0.7:



Hình 23. Confusion Matrix – "Linear" SVM (C=0.7)

5.4 Nhận xét các mô hình học máy

Model	Naïve Bayes	Logistic Regession	RBF SVM (C = 2)	Linear SVM (C = 0.7)
Train Accuracy	0.8866	0.9215	0.9908	0.9278
Valid Accuracy	0.8616	0.9006	0.9020	0.9015

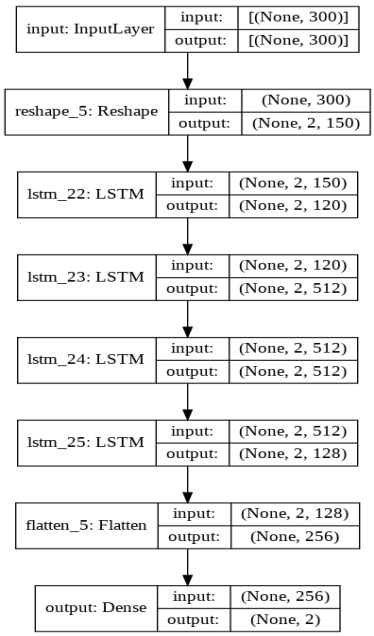
Từ bảng tổng kết trên ta thấy rằng trong các mô hình học máy thì 2 mô hình SVM vẫn có kết quả tốt hơn so với các mô hình khác. Cụ thể Naïve Bayes là mô hình học tệ nhất trong khi "RBF" SVM đạt valid accuracy cao nhất, tuy nhiên chỉ cao hơn "Linear" SVM một chút.

Ngoài ra, với điểm số khá ổn như trên dù chỉ sử dụng các mô hình phân loại tuyến tính (không tính "RBF" SVM), em nhận thấy rằng dữ liệu sau khi nhúng đã đạt được một mức độ tách biệt tuyến tính khá lớn giữa các class dữ liệu.

5.5 Long Short Term Memory (LSTM)

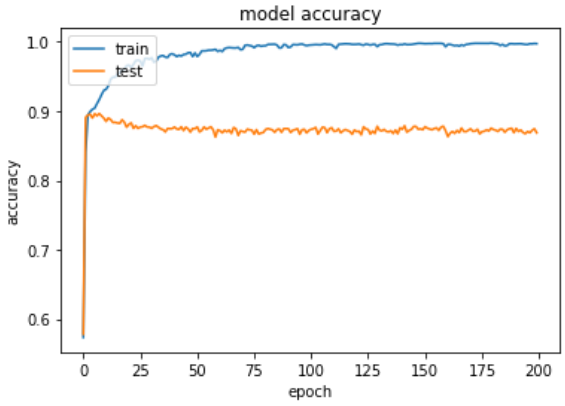
Để giảm thời gian tính toán của quá trình huấn luyện và kiểm thử. Thuật toán Singular Value Decomposition được sử dụng nhằm mục đích giảm chiều của ma trận mà em thu được sau khi mã hóa dữ liệu đầu vào theo phương pháp TF-IDF vectorize mà vẫn giữ được các thuộc tính của giữ liệu ban đầu.

Kiến trúc mô hình:

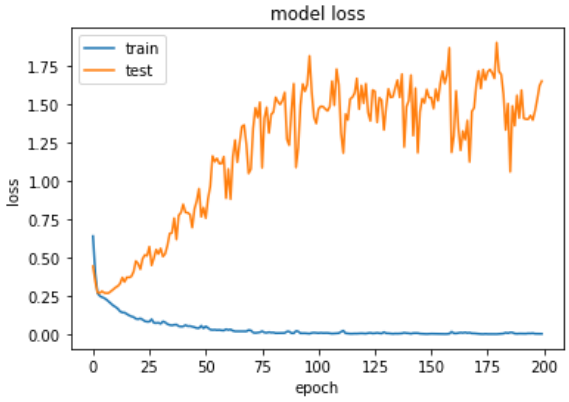


Hình 24. Kiến trúc mô hình LSTM

Huấn luyện và kiểm thử dữ liệu sau 200 epochs thu được kết quả đánh giá trên cả 2 tập huấn luyện và kiểm thử như sau:



Hình 25. Train và Valid Accuracy(LSTM)



Hình 26. Train và Valid Loss(LSTM)

Kết quả tốt nhất mô hình đạt được trên tập kiểm thử là 0.8964 vẫn còn kém hơn so với kết quả đạt được của mô hình học máy SVM và Logistic Regression. Điều này có thể giải thích là vì dữ liệu dùng cho quá trình huấn luyện chưa nhiều vậy nên kết quả của mô hình học sâu mang lại vẫn chưa tốt bằng những mô hình học máy thông thường.

5.6 PhoBERT

Sau khi chuẩn bị xong dữ liệu và tạo dataloader thành công ở bước tiền xử lý dữ liệu thì tiếp theo sẽ là quá trình huấn luyện và kiểm thử bộ dữ liệu đó bằng mô hình phoBERT.

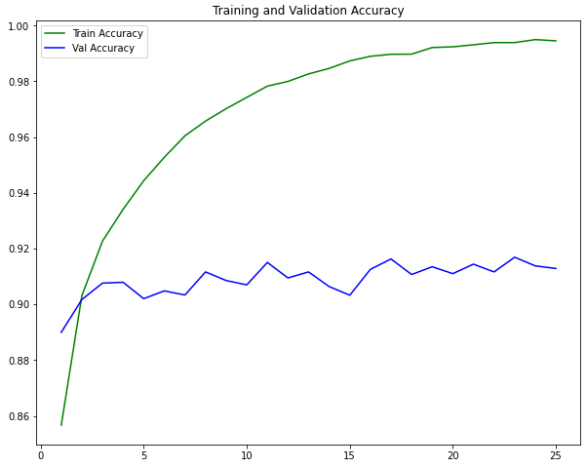
Sử dụng hàm RobertaConfig của thư viện transformer để load cấu hình của mô hình PhoBERT đã tải về, sau đó sử dụng hàm RobertaForSequenceClassifcation để load lại mô hình phoBERT cùng với cấu hình đã được load trước đó.

RobertaForSequenceClassification là mô hình BERT thông thường với một lớp tuyến tính duy nhất được thêm vào ở trên để phân loại, trong bài toán này nó sẽ được sử dụng để phân loại câu. Khi người dùng cung cấp dữ liệu đầu vào, toàn bộ mô hình BERT pre-trained và lớp phân loại sẽ được huấn luyện với nhiệm vụ cụ thể của bài toán.

```
from transformers import RobertaForSequenceClassification, RobertaConfig, AdamW

config = RobertaConfig.from_pretrained(
    "/content/drive/My Drive/Đồ án 2 (Sentiment Analysis Vietnamese)/PhoBERT_base_transformers/config.json", from_tf=False, num_labels = 2, output_hidden_states=False,
)
BERT_SA = RobertaForSequenceClassification.from_pretrained(
    "/content/drive/My Drive/Đồ án 2 (Sentiment Analysis Vietnamese)/PhoBERT_base_transformers/model.bin",
    config=config
)
```

Tiếp theo sẽ là quá trình huấn luyện và kiểm thử. Kết quả thu được sau 25 epochs được đánh giá trên cả 2 tập trainning và validation sẽ được biểu diễn trong biểu đồ dưới đây:



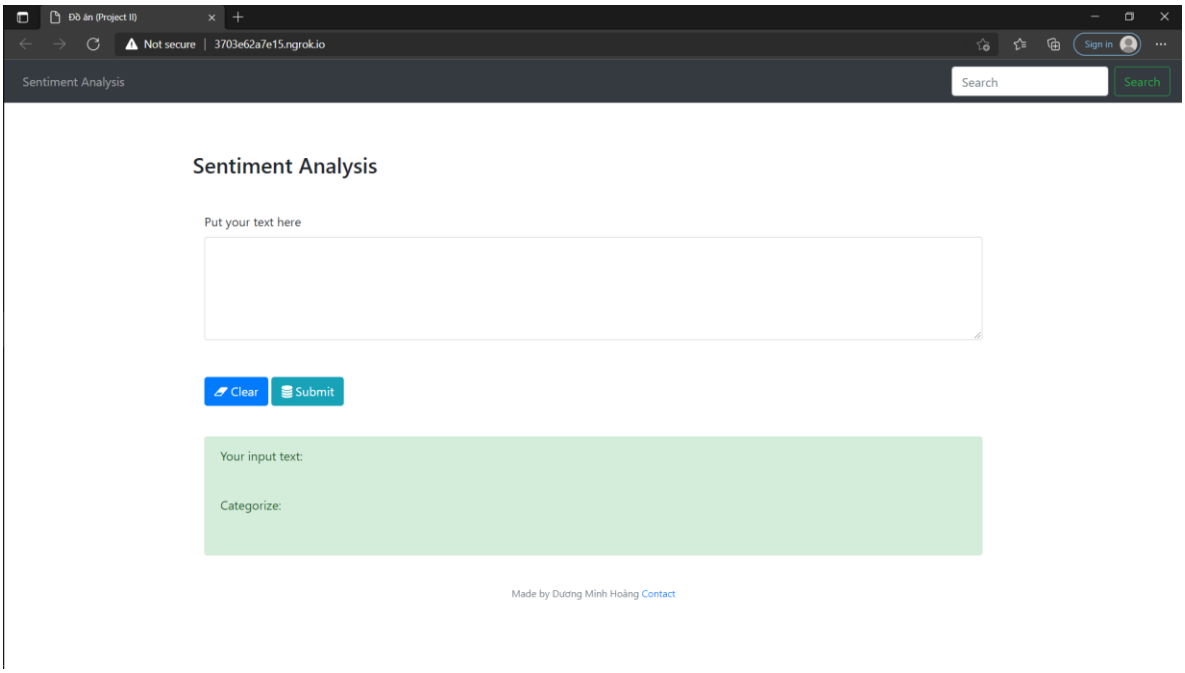
Hình 27. Train và Valid Accuracy(BERT) *Hình 28. Train và Valid Loss (BERT)*

Từ 2 biểu đồ trên ta có thể thấy được đã có hiện tượng overfit xảy ra trong quá trình học của máy khi sau khoảng 5 epochs thì quá trình học hầu như chỉ diễn ra trên tập train mà không diễn ra trên tập valid khiến cho giá trị loss của valid tăng sau mỗi epochs và giá trị Accuracy vẫn dao động quanh mức cũ. Điều này có thể giải thích là do phân bố giá trị trên tập train và tập valid chưa đều nên hiện tượng overfit đã xảy ra.

Tuy nhiên mô hình BERT vẫn là mô hình mang lại kết quả tốt nhất trong những mô hình được thử nghiệm ở trong bài toán này của em với kết quả đạt được trên tập valid là 0.9169

6. Demo chương trình

Sau quá trình huấn luyện và kiểm thử hoàn thành, em đã xây dựng giao diện cơ bản để có thể trực quan quá trình nghiệp vụ của bài toán yêu cầu. Sau đây là một vài kết quả của mô hình tốt nhất và giao diện ban đầu của demo.



Hình 29. Giao diện hệ thống

Your input text:

Tệ😡 Sản phẩm đứt chỉ tùm lum😡 Rách quá trời chỗ hàng lũng😡 Shop phục vụ quá tệ😡 Lần cuối mua shop😡

Categorize:

Tiêu cực

Your input text:

Dù rep ib hơi chậm nhưng chất lượng sản phẩm khá tốt. Không khác gì ảnh trong quảng cáo, tuy nhiên kim đồng hồ không có dạ quang đâu Nhưng nhìn rất sang cãm nặng trịch mình ưng ý

Categorize:

Tích cực

Hình 30. Một số kết quả demo chương trình

Source Code và Hướng dẫn cài đặt sản phẩm demo có sẵn tại:

https://github.com/hoangIT1/Sentiment-Analysis-Vietnamse.

7. Tổng kết:

Với đề tài lựa chọn là phân tích cảm xúc trong Tiếng Việt (Sentiment Analysis Vietnamese) là một bài toán không quá xa lạ trong lĩnh vực xử lý ngôn ngữ tự nhiên em đã hoàn thành tương đối việc đánh giá các mô hình từ học máy đến học sâu và kết quả đạt được tốt nhất hiện tại của em là 0.9169 đối với mô hình BERT - một kết quả tạm chấp nhận được. Tuy nhiên trong quá trình học vẫn diễn ra hiện tượng overfit khiến cho mô hình học được chủ yếu là ở tập train ở các epochs sau mà không có ảnh hưởng quá nhiều đến tập valid. Vì thế trong tương lai hướng cải tiến của em vẫn sẽ tiến hành trên mô hình BERT nhằm cải thiện hiện tượng overfit và đem lại kết quả đáng mong đợi hơn cho bài toán.

8. Tài liệu tham khảo

[1] AIVIVN 2019 Contest: Sentiment Analysis Challenge. [Online]. Available: https://www.aivivn.com/contests/1.

[2] Vi Ngo Van, Minh Hoang Van, Tam Nguyen Thanh, Sentiment Analysis for Vietnamese using Support Vector Machine with application to Facebook comments.

[3] Dat Quoc Nguyen, Anh Tuan Nguyen, PhoBERT: Pre-trained language models for Vietnamese.

[4] Minh-Hoang Bui, Hướng dẫn chi tiết về cơ chế của LSTM và GRU trong NLP. [Online]. Available: https://blog.chappiebot.com/hướng-dẫnchi-tiết-về-cơ-chế-của-lstm và-gru-trong-nlp-a1bd9346b209.

[5] Quynh-Trang Thi Pham, Xuan-Truong Nguyen, Thi-Cham Nguyen, Mai-Vu Tran, DSKTLAB: Vietnamese Sentiment Analysis for Product Reviews.

[6] Phạm Hữu Quang, BERT, RoBERTa, PhoBERT, BERTweet: Ứng dụng state-of-the-art pre-trained model cho bài toán phân loại văn bản. [Online]. Available: https://viblo.asia/p/bert-roberta-phobert-bertweet-ung-dung-state-of-the-art-pre-trained-model-cho-bai-toan-phan-loai-van-ban-4P856PEWZY3.

[7] Manish Munikar, Sushil Shakya, Aakash Shrestha, Fine-grained Sentiment Classification using BERT.