

PROJECT 03

Phân tích dữ liệu trên web

1. Thông tin thành viên

STT	Thành viên 1	Thành viên 2
MSSV	20424008	20424013
Họ và tên	Đương Mạnh Cường	Phạm Nguyễn Mỹ Diễm
Email	20424008@student.hcmus.edu.vn	20424013@student.hcmus.edu.vn

2. Phân công công việc và quá trình thực hiện

2.1 Phân công công việc

STT	Công việc		Thành viên thực hiện	Thời gian thực hiện (Deadline: ~ 4 tuần)	Hoàn thành của thành viên (%)	Hoàn thành tổng thể (%)
1	Chia bộ dữ liệu thành hai phần training data và test data nhằm đảm bảo tính công bằng trên mọi model.		Cường – Diễm	Tuần 1	100	100
2	Chia bài toán ban đầu thành hai bài toán con là Emoji sentiment model và Comment sentiment model và giải thích lí do.	Xây dựng Emoji sentiment model .	Cường – Diễm	Tuần 2 – 4	100	100
		Xây dựng Comment sentiment model	Cường – Diễm	Tuần 2 - 4	100	100
4	Tổng hợp hai model Emoji sentiment và Comment sentiment để ra model cuối cùng.		Cường – Diễm	Tuần 2 – 4	100	100
5	Viết báo cáo.		Cường – Diễm	Tuần 4	100	100

2.2 Tóm tắt quá trình thực hiện

Ở project 3 bài chúng ta sẽ lần lượt làm các công việc sau:

- **Công việc 1:** Chia bộ dữ liệu thành hai phần training data và test data nhằm đảm bảo tính công bằng trên mọi model.
- **Công việc 2:** Chia bài toán ban đầu thành hai bài toán con là **Emoji sentiment model** và **Comment sentiment model** và giải thích lí do.
 - **Công việc 2.1:** Xây dựng **Emoji sentiment model**.

- Xác định input và cách biểu diễn nó sau đó lựa chọn cách biểu diễn phù hợp.
- Xác định output.
- Tiến hành sử dụng các **Traditional Machine Learning Classifier** để đào tạo.
- Tối ưu hóa tham số bằng phương pháp **Grid Search**.
- Đánh giá model.
- Lưu lại model.
- Công việc 2.2: Xây dựng **Comment sentiment model**.
 - Xác định input và cách biểu diễn nó sau đó lựa chọn cách biểu diễn phù hợp.
 - Xác định output.
 - Tiến hành sử dụng các **Traditional Machine Learning Classifier** để đào tạo.
 - Tiến hành sử dụng các **Deep Learning** để đào tạo.
 - Tối ưu hóa tham số bằng phương pháp **HyperBand**.
 - Đánh giá model.
 - Lưu lại model.
- **Công việc 3:** Tổng hợp hai model **Emoji sentiment** và **Comment sentiment** để ra model cuối cùng.

3. Báo cáo theo các mục ở phần 2.1

3.1 Công việc 1 (File: 05.model.ipynb)

- Phần này ta sẽ tách dữ liệu sau khi trải qua các bước tiền xử lí ở project 1 thành training data và test data.
- Lí do ta cần thực hiện điều này là ta muốn đảm bảo công bằng cho mọi model trong quá trình đào tạo, tức chúng cùng học trên cùng một training data và được đánh giá trên cùng một test data.
- Như ở **công việc 2** đã trình bày, ta sẽ chia dữ liệu sau tiền xử lí thành hai phần:
 - **Phần dữ liệu chỉ chứa emoji:** phần dữ liệu này chỉ chứa các comment chứa emoji, các comment không chứa emoji ta sẽ loại bỏ.
 - **Phần dữ liệu chỉ chứa comment:** phần dữ liệu này chính là phần dữ liệu ban đầu nhưng khác một điều toàn bộ emoji trong comment sẽ bị xóa.
- Đọc toàn bộ dữ liệu sau khi đã tiền xử lí ở project 1.

	raw_comment	normalize_comment	label
0	Giao hàng kh đúng cần phê bình hjjjjjhhd...	giao hàng không đúng cần phê bình	0
1	Chất lượng sản phẩm tạm được. Giao...	chất lượng sản phẩm tạm được giao ...	0
2	Ko có lắc tay như hình	không có lắc tay như hình	0
3	Giao hàng lâu. Bảo có lắc tay mà k thâ...	giao hàng lâu bảo có lắc tay mà không ...	0
4	Mình mua 2 cái, một dùng ok. Một cái k...	mua cái một dùng ok một cái không chạ...	0

- Mã hóa các dữ liệu dạng text về cùng một dạng là **NFD**.

```

1 data = Model.textNFDxformat(data, [ 'raw_comment', 'normalize_comment'], 'NFD')
2
3 data.head()

```

	raw_comment	normalize_comment	label
0	Giao hàng kh đúng cần phê bình haaaaahd...	giao hàng không đúng cần phê bình	0
1	Chất lượng sản phẩm tạm được. Giao...	chất lượng sản phẩm tạm được giao ...	0
2	Ko có lắc tay như hình	không có lắc tay như hình	0
3	Giao hàng lâu. Bảo có lắc tay mà k thâ...	giao hàng lâu bảo có lắc tay mà không ...	0
4	Mình mua 2 cái, một dùng ok. Một cái k...	mua cái một dùng ok một cái không chạ...	0

- Các vectorizer object của **sklearn** mặc định chúng sẽ xóa toàn bộ các **punctuation** [kí tự đặc biệt] trong input truyền vào. Như vậy các emoji của ta sẽ bị xóa toàn bộ khi transform vectorizing. Như vậy, ta sẽ không lưu chúng dưới dạng các punctuation mà dùng decode của chúng - sau đó ta tạo feature **emoji_decode** để lưu chúng.

```

1 data['emoji_decode'] = data['raw_comment'].apply(lambda s: Model.expandEmojisDecode(s))
2 data = data[['raw_comment', 'normalize_comment', 'emoji_decode', 'label']]
3
4 data.head()

```

	raw_comment	normalize_comment	emoji_decode	label
0	Giao hàng kh đúng cần phê bình haaaaahd...	giao hàng không đúng cần phê bình		0
1	Chất lượng sản phẩm tạm được. Giao...	chất lượng sản phẩm tạm được giao ...		0
2	Ko có lắc tay như hình	không có lắc tay như hình		0
3	Giao hàng lâu. Bảo có lắc tay mà k thâ...	giao hàng lâu bảo có lắc tay mà không ...		0
4	Mình mua 2 cái, một dùng ok. Một cái k...	mua cái một dùng ok một cái không chạ...	cry	0

- Tiến hành chọn các mẫu có feature **emoji_decode** không phải là chuỗi rỗng và lưu vào biến **emoji_data**. Các mẫu trong biến này sẽ được dùng để xây dựng một **Emoji sentiment model**.

```

1 emoji_data = data[data['emoji_decode'] != ""].reset_index(drop=True)
2
3 emoji_data.head()

```

	raw_comment	normalize_comment	emoji_decode	label
0	Mình mua 2 cái, một dùng ok. Một cái k...	mua cái một dùng ok một cái không chạ...		cry 0
1	Giao sai màu sai size có 1 dép lông size 3...	giao sai màu sai size có dép lông size à ...	cursing_face cursing_face cursing_face	0
2	Đơn hàng đã thanh toán airpay rồi mà sh...	đơn hàng thanh toán mà giao cho người l...	roll_eyes	0
3	Áo croptop freesize rộng với mình \nMin...	áo rộng mình kg thumbsup thumbsup thumbsup thumbsup thumbsup thumbsup t...		0
4	Khá buồn . Đặt 2 cái đều không chạy ...	khá buồn đặt cái đều không chạy giờ...	female_sign shrug	0

- Giờ thì ta sẽ tiến hành chia **emoji_data** thành training data và test data với size của test data là 20%, sau đó ta lưu chúng dưới dạng file **.csv**.

```
1 Model.dataSplitSaved(emoji_data, 0.2, "./data/emoji_data")
```

🔊 Your dataset has saved at ./data/emoji_data.

- Bây giờ ta cần chuẩn bị training data và test data cho **Comment sentiment model**, ta cũng sẽ chia tập dữ liệu sau tiền xử lí thành hai phần training data và test data với test data chiếm 20% dữ liệu sau tiền xử lí.
- Cuối cùng ta cũng sẽ lưu hai tập training data và test data này dưới dạng file .csv.

```
1 data.head()
```

	raw_comment	normalize_comment	emoji_decode	label
0	Giao hàng kh ^d ng c ^d n ph ^e b ^e h ^j jjjjhh...	giao hàng kh ^d ng c ^d n ph ^e b ^e h ^j ...		0
1	Ch ^a t l ^a ng s ^a n p ^a nh t ^a m đ ^a ng c ^a o. Giao...	ch ^a t l ^a ng s ^a n p ^a nh t ^a m đ ^a ng giao ...		0
2	Ko có l ^a ć tay nh ^a h ^a nh	kh ^a ng có l ^a ć tay nh ^a h ^a nh		0
3	Giao hàng l ^a u. B ^a o có l ^a ć tay m ^a k th ^a ...	giao hàng l ^a u b ^a o có l ^a ć tay m ^a kh ^a ng ...		0
4	Mình mua 2 cái, m ^a t d ^a ng ok. M ^a t cái k...	mua cái m ^a t d ^a ng ok m ^a t cái kh ^a ng ch ^a ...	cry	0

```
1 Model.dataSplitSaved(data, 0.2, "./data/data")
```

🔊 Your dataset has saved at ./data/data.

3.2 Công việc 2. (File: 05.model.ipynb)

- Ở phần này, nhóm sẽ trình bày về các vấn đề sau:

- **Vấn đề 1:** Lựa chọn thuật toán tương ứng lần lượt cho hai model và lí giải.
- **Vấn đề 2:** Lựa chọn kĩ thuật đánh giá.

- **Vấn đề 1:**

- Bài toán của chúng ta là NLP - như vậy thì dữ liệu sẽ khiến cho chúng ta khó hiểu hơn về dữ liệu. Nhưng theo những gì nhóm đã được học ở những môn trước thì nhóm có hi vọng cao vào hai thuật toán là **Logistic Regression** và **Support Vector Machine**. Tuy nhiên nhóm vẫn sẽ áp dụng các model classification khác như Naive Bayes, Random Forest,... vì khả năng cao chúng có thể đại diện tốt cho dataset của chúng ta.

- **Logistic Regression**

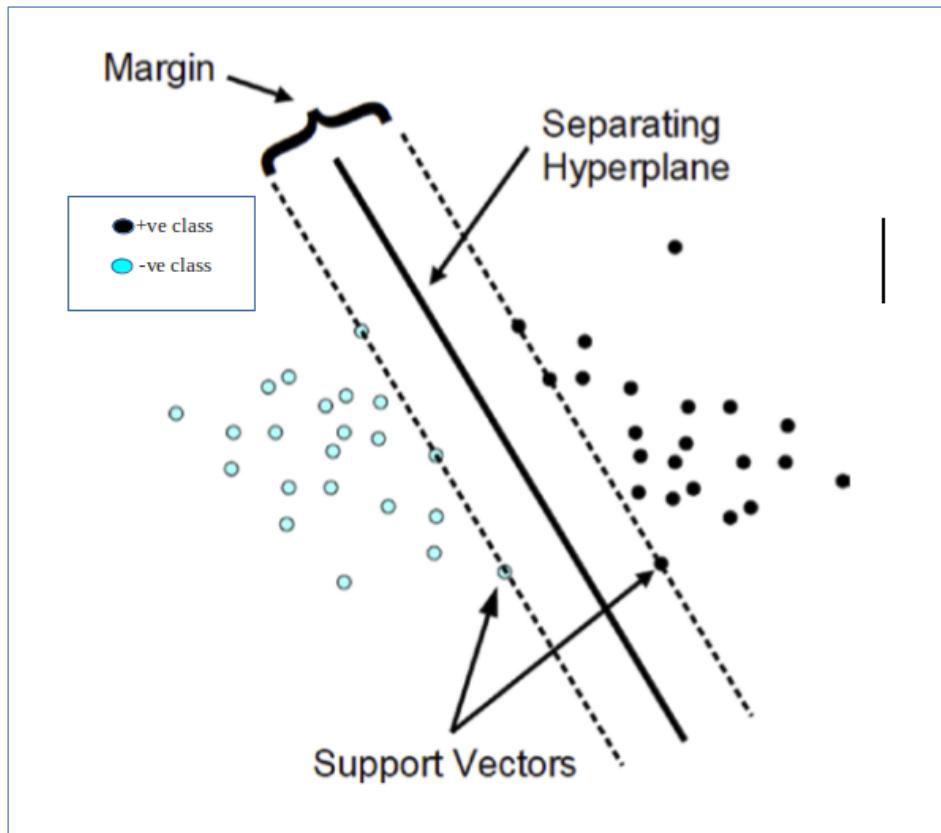
- Ở phần đào tạo mô hình sau này, nhóm sẽ ưu tiên sử dụng thuật toán này trên các solver khác nhau như **newton-cg**, **lbfgs**, **liblinear**. Đây là một sự ưu tiên cho thuật toán này vì nhóm nghĩ nó hiệu quả vì:

- Bài toán của chúng ta là **binary classification** và **Logistic Regression** thường được coi là thuật toán cơ bản nhất cho các bài toán dạng này.
- Thuật toán **Logistic Regression** có thời gian thực thi nhanh và cách cài đặt đơn giản, các **hyper-parameter** không nhiều nên dễ dàng thực hiện kỹ thuật **Tunning Hyper-Parameters**.
- Đối với **Emoji sentiment model**, thực chất số lượng emoji mà người dùng hay dùng không nhiều, số emoji trong một comment cũng không nhiều \Rightarrow Khiến cho dữ liệu đào tạo đơn giản và dễ hiểu nên **Logistic Regression** rất phù hợp với các dataset đơn giản như vậy đồng thời sẽ cho ra độ chính xác cao.
- Với **Comment sentiment model** - dữ liệu phức tạp hơn nhưng chúng ta cũng nên kì vọng là thuật toán này sẽ hoạt động tốt.

- **Support Vector Machine**

- Do nhóm nghĩ đây là một thuật toán hiệu quả, nên nhóm cũng sẽ có chút ưu tiên cho thuật toán bày bằng cách triển khai nó trên nhiều **kernel** khác nhau như **linear**, **poly**, **rbf**, **sigmoid**. Lý do nhóm ưu tiên thuật toán này là vì:

- Với các dữ liệu mà ta khó có cái nhìn tổng quan hoặc ý tưởng thì SVM là một mô hình khá tốt để ta tiến hành đào tạo vì nó linh hoạt - có thể dùng cho hai bài toán là **regression** và **classification** thậm chí là cho cả các bài toán **clustering**.
- Nó hoạt động tốt trên dữ liệu phức tạp mà với dữ liệu text thì text hay được biểu diễn dưới dạng vector.
- Với các bài toán phân lớp, nó sử dụng các **kernel** để đưa input đầu vào vào một không gian có nhiều chiều hơn ngoài ra còn cố gắng tối đa hóa khoảng cách giữa **separating hyperplane** với các **super vectors**.



- Hoạt động hiệu quả trên bài toán phân loại văn bản, dữ liệu phi cấu trúc và nhiều chiều.
- Ngoài ra, sức mạnh của thuật toán này chính là dựa trên các **kernel** mà ta lựa chọn, tuy nhiên để chọn ra **kernel** tốt không dễ dàng nên ta thường vét cạn, nhưng nếu dữ liệu quá lớn thì không nên vì thời gian đào tạo của thuật toán này lâu, có thể nói ngang ngửa với **các Deep Neural Network**.

- **Deep Neural Network**

- Các thuật toán thuộc nhóm DNN sẽ được trình bày sau. Nhóm sẽ tập trung trình bày vào LSTM vì đây là model hoạt động tốt nhất trên dataset của nhóm.

- **Vấn đề 2:**

		Predicted Class	
		Class 0 (negative)	Class 1 (positive)
Actual Class	Class 0 (negative)	TN	FP (Type-I Error)
	Class 1 (positive)	FN (Type-II Error)	TP

- Chúng ta sẽ sử dụng hai độ đo phổ biến nhất dành cho các **classification model** là:
 - **Accuracy**: dùng để đánh giá độ chính xác của model trên TN và TP.
 - **ROC-AUC**: accuracy sẽ không chính xác nếu như số lượng mẫu giữa các class bị mất cân bằng nên ROC-AUC sẽ giúp ta kiểm tra việc xem có một class nào nổi trội hơn so với class còn lại không.

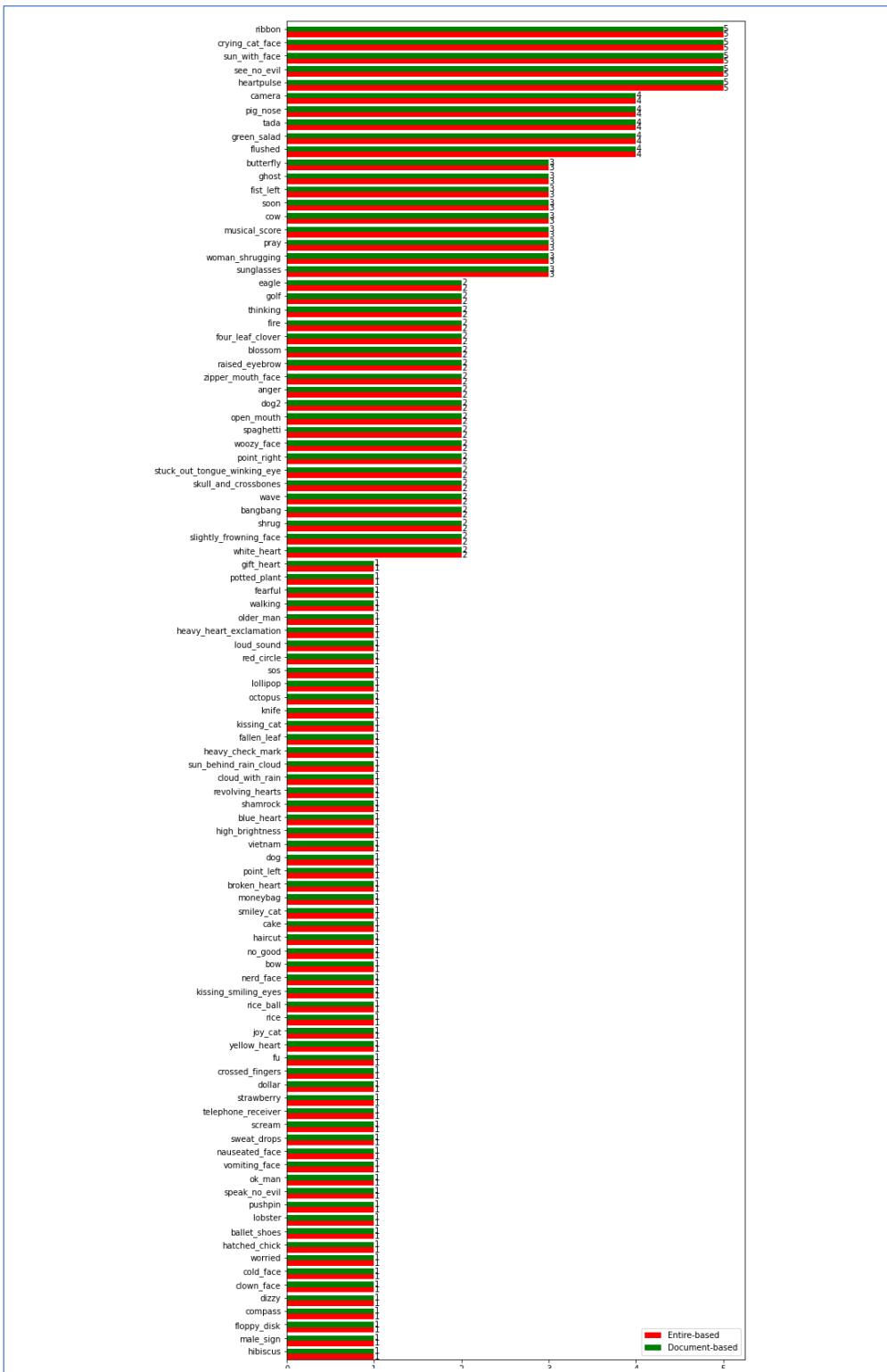
3.2.1 Công việc 2.1 (File 06.model.ipynb)

- Phần này, ta sẽ đào tạo một **emoji sentiment model**.
- Chiến lược của ta sẽ như sau:
 - **Bước 1**: Ta tiến hành định nghĩa một vài vectorizer bằng **sklearn**.
 - **Bước 2**: Liệt kê các classifier model.
 - **Bước 3**: Với từng vectorizer đã liệt kê, sử dụng từng model classifier để đào tạo cross-validation với input là vectorizer tương ứng.
 - **Bước 4**: Lựa chọn các model và vectorizer phù hợp với dataset emoji.
 - **Bước 5**: Thực hiện tuning hyperparams cho các model được chọn ở bước 4 bằng cross-validation.
 - **Bước 6**: Sau khi tuning và nhận được các hyperparams tốt nhất, tiến hành training lại nhưng không cross-validation.
 - **Bước 7**: Lưu lại các model này thành file ***.pickle**.
 - **Bước 8**: Đánh giá bằng accuracy và ROC-AUC trên test data.

- Đọc training data.

1	<code>X_train, y_train = Model.loadData("./data/emoji_data/train")</code>		
1	<code>display(X_train.head(), y_train.head())</code>		
<hr/>			
	raw_comment	normalize_comment	emoji_decode
0	Cái 9k đẹp hơn.Cái 9k kiểu bóng trông ...	cái không đẹp hơn cái không kiểu bóng...	heart_eyes
1	Mũ xinh hihih chờ hàng hơi lâu tí nhưng...	mũ xinh chờ hàng hơi lâu tí nhưng nhạ...	hearts hearts hearts kissing_heart kissing_he...
2	Thời gian giao hàng rất chậm !\nHàng g...	thời gian giao hàng rất chậm hàng giao...	relieved
3	Áo không biết là mới hay k á, nhưng m...	áo không biết là mới hay không á như...	sob sob sob persevere persevere persevere pers...
4	Da mềm êm rất đẹp nhé❤cúc bấm cung...	da mềm êm rất đẹp nhé cúc bấm cung ...	heart heart
label			
0	0		
1	1		
2	0		
3	1		
4	1		

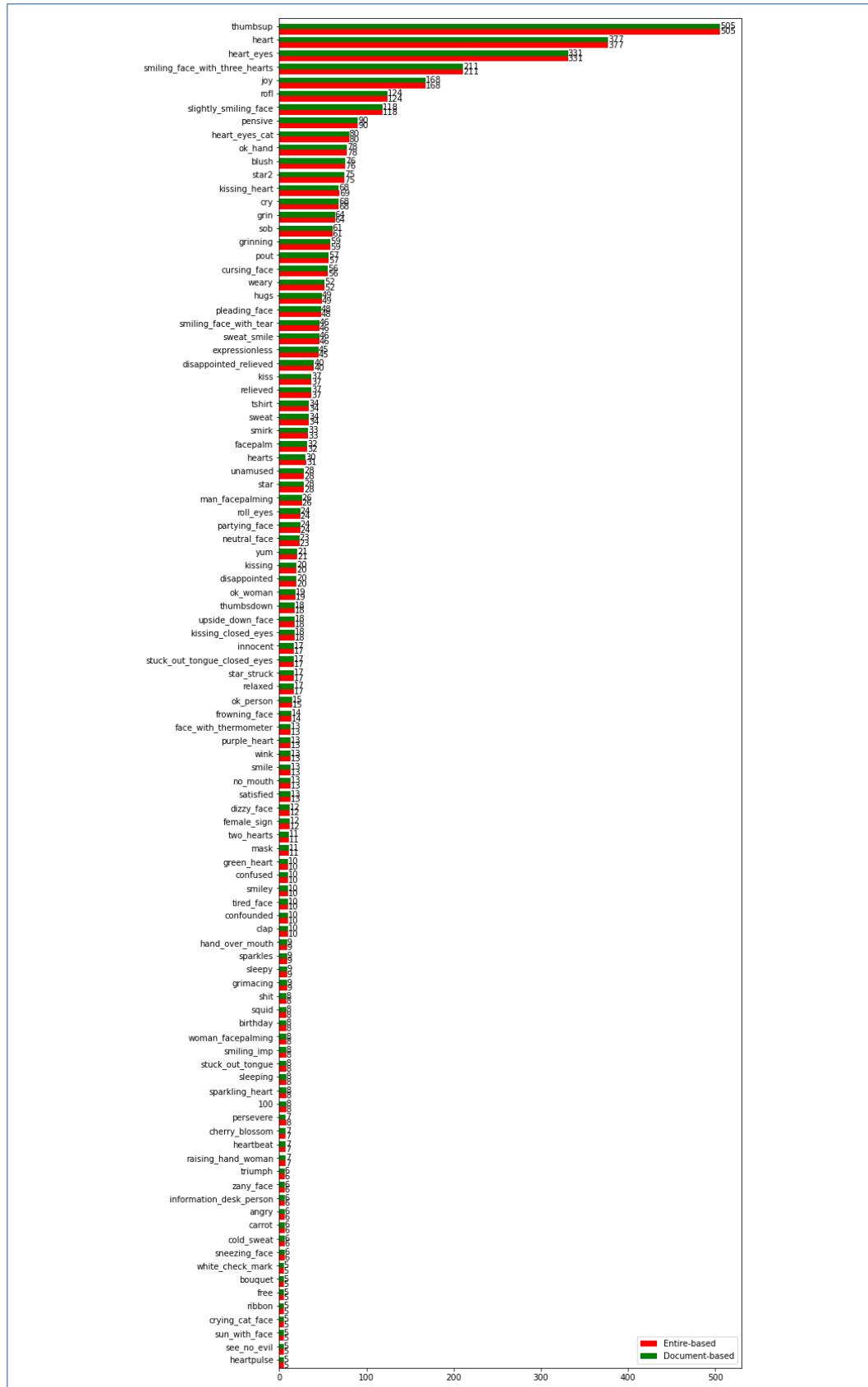
- Hãy xem lại các emoji của chúng ta phân bổ như thế nào, thứ mà chúng ta đã không quan tâm lăm ở project 2.
- Sau bước này, chúng ta sẽ tiến hành vectorizer cho các emoji được tách ra từ các comment.



- Nhận xét:

- Biểu đồ trên là 100 emoji có tần số xuất hiện thấp nhất trên toàn dataset emoji.
- Ở đây, các thanh màu đỏ thể hiện tần suất của từ đó xuất hiện bao nhiêu lần trong dataset. Các thanh màu xanh thể hiện có bao nhiêu comment chứa emoji này.

- Có thể thấy có nhiều emoji mà chỉ xuất hiện một lần. Như vậy ta có thể xem xét xây dựng các vectorizer với các ngưỡng **min_df** (*được nói chi tiết bên dưới*) khác nhau để tìm xem đâu là vectorizer phù hợp nhất để làm input cho dataset emoji này.



- **Nhận xét:**

- Bên trên là 100 emoji có tần số xuất hiện nhiều nhất.
 - Từ hai biểu đồ trên, có thể tạo ra các vectorizer mà chúng chứa toàn bộ các emoji và các emoji có tần số xuất hiện ≥ 5 lần.
- Dưới đây, ta xây dựng bốn vectorizer theo các phương pháp **Bag of Words** và **TF-IDF**. Và lưu chúng vào một **list**.
- Hai vectorizer đầu tiên, ta sử dụng toàn bộ các emoji trong dataset emoji.
- Hai vectorizer cuối cùng, ta chỉ vectorize cho các emoji mà có tần số xuất hiện $\geq (\text{min_df}=5)$.

```

1 vectorizers = [
2     ("Bag of Words", Model.vectorizer(X_train['emoji_decode'], 'bow')),
3     ("TF-IDF", Model.vectorizer(X_train['emoji_decode'], 'tfidf')),
4     ("Bag of Words - [min_df: 5]", Model.vectorizer(X_train['emoji_decode'], 'bow', 5)),
5     ("TF-IDF - [min_df: 5]", Model.vectorizer(X_train['emoji_decode'], 'tfidf', 5))
6 ]

```

- Ta tiến hành kiểm tra vectorizer đầu tiên - Bag of Words trên toàn bộ dataset emoji.

```

['100', 'anger', 'angry', 'ballet_shoes', 'bangbang', 'birthday', 'blossom', 'blue_heart', 'blush', 'bouquet', 'bow', 'broken_heart', 'butterfly', 'cake', 'camera', 'carrot', 'cherry_blossom', 'clap', 'cloud_with_rain', 'clown_face', 'cold_face', 'cold_sweat', 'compass', 'confounded', 'confused', 'cow', 'crossed_fingers', 'cry', 'crying_cat_face', 'cursing_face', 'disappointed', 'disappointed_relieved', 'dizzy', 'dizzy_face', 'dog', 'dog2', 'dollar', 'eagle', 'expressionless', 'face_with_thermometer', 'facepalm', 'fallen_leaf', 'fearful', 'female_sign', 'fire', 'fist_left', 'floppy_disk', 'flushed', 'four_leaf_clover', 'free', 'frowning_face', 'fu', 'ghost', 'gift_heart', 'golf', 'green_heart', 'green_salad', 'grimacing', 'grin', 'grinning', 'haircut', 'hand_over_mouth', 'hatched_chick', 'heart', 'heart_eyes', 'heart_eyes_cat', 'heartbeat', 'heartpulse', 'hearts', 'heavy_check_mark', 'heavy_heart_exclamation', 'hibiscus', 'high_brightness', 'hugs', 'information_desk_person', 'innocent', 'joy', 'joy_cat', 'kiss', 'kissing', 'kissing_cat', 'kissing_closed_eyes', 'kissing_heart', 'kissing_smiling_eyes', 'knife', 'lobster', 'lollipop', 'loud_sound', 'male_sign', 'man_facepalming', 'mask', 'moneybag', 'musical_score', 'nauseated_face', 'nerd_face', 'neutral_face', 'no_good', 'no_mouth', 'octopus', 'ok_hand', 'ok_man', 'ok_person', 'ok_woman', 'older_man', 'open_mouth', 'partying_face', 'pensive', 'persevere', 'pig_nose', 'pleading_face', 'point_left', 'point_right', 'potted_plant', 'pout', 'pray', 'purple_heart', 'pushpin', 'raised_eyebrow', 'raising_hand_man', 'red_circle', 'relaxed', 'relieved', 'revolving_hearts', 'ribbon', 'rice', 'rice_ball', 'rofl', 'roll_eyes', 'satisfied', 'scream', 'see_no_evil', 'shamrock', 'shit', 'shrug', 'skull_and_crossbones', 'sleeping', 'sleepy', 'slightly_frowning_face', 'slightly_smiling_face', 'smile', 'smiley', 'smiley_cat', 'smiling_face_with_tear', 'smiling_face_with_three_hearts', 'smiling_imp', 'smirk', 'sneezing_face', 'sob', 'soon', 'sos', 'spaghetti', 'sparkles', 'sparkling_heart', 'speak_no_evil', 'squid', 'star', 'star2', 'star_struck', 'strawberry', 'stuck_out_tongue', 'stuck_out_tongue_closed_eyes', 'stuck_out_tongue_winking_eye', 'sun_behind_rain_cloud', 'sun_with_face', 'sunglasses', 'sweat', 'sweat_drops', 'sweat_smile', 'tada', 'telephone_receiver', 'thinking', 'thumbsdown', 'thumbsup', 'tired_face', 'triumph', 'tshirt', 'two_hearts', 'unamused', 'upside_down_face', 'vietnam', 'vomiting_face', 'walking', 'wave', 'weary', 'white_check_mark', 'white_heart', 'wink', 'woman_facepalming', 'woman_shrugging', 'woozy_face', 'worried', 'yellow_heart', 'yum', 'zany_face', 'zipper_mouth_face']

```

100	anger	angry	ballet_shoes	bangbang	birthday	blossom	blue_heart	blush	bouquet	...	white_heart	wink	womar
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0
...
1021	0	0	0	0	0	0	0	0	0	0	0	0	0
1022	0	0	0	0	0	0	0	0	0	0	0	0	0
1023	0	0	0	0	0	0	0	0	0	0	0	0	1
1024	0	0	0	0	0	0	0	0	11	0	0	0	0
1025	0	0	0	0	0	0	0	0	0	0	0	0	1

1026 rows × 195 columns

- **Nhận xét:** Ta có tổng cộng gần 200 emoji khác nhau trên hơn 1000 comment trong training dataset. Có thể thấy đây là một sparse-matrix.

100	anger	angry	ballet_shoes	bangbang	birthday	blossom	blue_heart	blush	bouquet	...	white_heart	wink	wo
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
...
1021	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
1022	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
1023	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.713115
1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.000000
1025	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.000000

1026 rows × 195 columns

- **Nhận xét:** Phía trên ta kiểm tra vectorizer TF-IDF trên toàn bộ dataset emoji.
- Vì dữ liệu ta là các vector 195 dim, ta rất khó để chọn ra một thuật toán phù hợp bằng các kĩ thuật trực quan hóa hay thống kê ban đầu.
- Vậy nên, ta sẽ liệt kê một loạt các classification model, để chuẩn bị trước cho quá trình đào tạo.
- Tiếp theo, ta sẽ áp dụng cross-validation, và ta muốn đảm bảo tính công bằng cho toàn bộ model nên ta sẽ tạo một đối tượng **StratifiedKFold**, như code bên dưới, nó chia training data thành 10 phần và giữ các thứ tự đó trong toàn bộ vòng đời của biến chứa nó.

```
1 lst_models = [
2     ('Logistic Regression - [solver: lbfgs]', LogisticRegression(solver='lbfgs')),
3     ('Logistic Regression - [solver: liblinear]', LogisticRegression(solver='liblinear')),
4     ('Logistic Regression - [solver: newton-cg]', LogisticRegression(solver='newton-cg')),
5     ('KNN - [n_neighbors: 2]', KNeighborsClassifier(n_neighbors=2)),
6     ('KNN - [n_neighbors: 3]', KNeighborsClassifier(n_neighbors=3)),
7     ('SVC - [kernel: linear]', SVC(kernel='linear', random_state=42)),
8     ('SVC - [kernel: poly]', SVC(kernel='poly', random_state=42)),
9     ('SVC - [kernel: rbf]', SVC(kernel='rbf', random_state=42)),
10    ('SVC - [kernel: sigmoid]', SVC(kernel='sigmoid', random_state=42)),
11    ('Bernoulli', BernoulliNB()),
12    ('Random Forest', RandomForestClassifier(random_state=42)),
13    ('XGBoost', XGBClassifier(eval_metric='mlogloss'))
14 ]
15
16 cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

- Đào tạo toàn bộ các model trên.

```
Bag of Words:
Model Logistic Regression - [solver: lbfgs] has been trained in 0.23 seconds
Model Logistic Regression - [solver: liblinear] has been trained in 0.06 seconds
Model Logistic Regression - [solver: newton-cg] has been trained in 0.28 seconds
Model KNN - [n_neighbors: 2] has been trained in 0.50 seconds
Model KNN - [n_neighbors: 3] has been trained in 0.51 seconds
Model SVC - [kernel: linear] has been trained in 0.28 seconds
Model SVC - [kernel: poly] has been trained in 0.42 seconds
Model SVC - [kernel: rbf] has been trained in 0.45 seconds
Model SVC - [kernel: sigmoid] has been trained in 0.36 seconds
Model Bernoulli has been trained in 0.06 seconds
Model Random Forest has been trained in 2.35 seconds
Model XGBoost has been trained in 0.84 seconds

TF-IDF:
Model Logistic Regression - [solver: lbfgs] has been trained in 0.15 seconds
Model Logistic Regression - [solver: liblinear] has been trained in 0.06 seconds
Model Logistic Regression - [solver: newton-cg] has been trained in 0.19 seconds
Model KNN - [n_neighbors: 2] has been trained in 0.51 seconds
Model KNN - [n_neighbors: 3] has been trained in 0.54 seconds
Model SVC - [kernel: linear] has been trained in 0.24 seconds
Model SVC - [kernel: poly] has been trained in 0.32 seconds
Model SVC - [kernel: rbf] has been trained in 0.33 seconds
Model SVC - [kernel: sigmoid] has been trained in 0.28 seconds
Model Bernoulli has been trained in 0.07 seconds
Model Random Forest has been trained in 2.18 seconds
Model XGBoost has been trained in 0.92 seconds
```

```

Bag of Words - [min_df: 5]:
Model Logistic Regression - [solver: lbfgs] has been trained in 0.17 seconds
Model Logistic Regression - [solver: liblinear] has been trained in 0.07 seconds
Model Logistic Regression - [solver: newton-cg] has been trained in 0.27 seconds
Model KNN - [n_neighbors: 2] has been trained in 0.56 seconds
Model KNN - [n_neighbors: 3] has been trained in 0.54 seconds
Model SVC - [kernel: linear] has been trained in 0.27 seconds
Model SVC - [kernel: poly] has been trained in 0.40 seconds
Model SVC - [kernel: rbf] has been trained in 0.41 seconds
Model SVC - [kernel: sigmoid] has been trained in 0.35 seconds
Model Bernoulli has been trained in 0.06 seconds
Model Random Forest has been trained in 1.75 seconds
Model XGBoost has been trained in 0.49 seconds

TF-IDF - [min_df: 5]:
Model Logistic Regression - [solver: lbfgs] has been trained in 0.13 seconds
Model Logistic Regression - [solver: liblinear] has been trained in 0.06 seconds
Model Logistic Regression - [solver: newton-cg] has been trained in 0.17 seconds
Model KNN - [n_neighbors: 2] has been trained in 0.49 seconds
Model KNN - [n_neighbors: 3] has been trained in 0.51 seconds
Model SVC - [kernel: linear] has been trained in 0.21 seconds
Model SVC - [kernel: poly] has been trained in 0.26 seconds
Model SVC - [kernel: rbf] has been trained in 0.29 seconds
Model SVC - [kernel: sigmoid] has been trained in 0.25 seconds
Model Bernoulli has been trained in 0.06 seconds
Model Random Forest has been trained in 1.64 seconds
Model XGBoost has been trained in 0.47 seconds

```

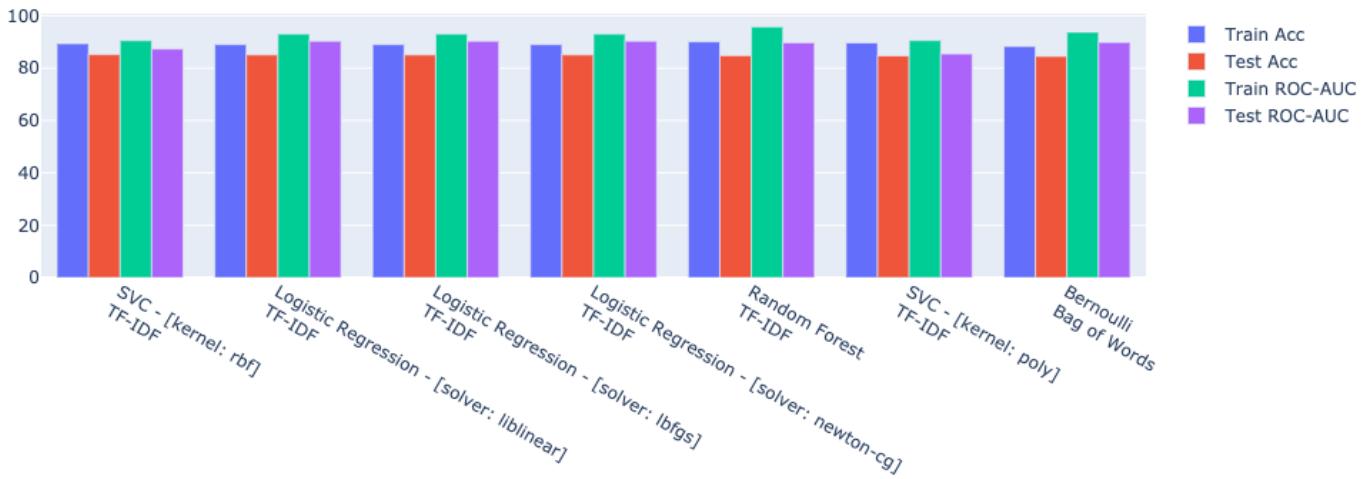
vectorizer	model	train_acc	test_acc	diff_acc	train_acc_std	test_acc_std	train_roc_auc	test_roc_auc	diff_roc_auc	
0	TF-IDF	SVC - [kernel: rbf]	0.893546	0.851875	0.041671	0.003487	0.038218	0.905123	0.873282	0.031841
1	TF-IDF	Logistic Regression - [solver: liblinear]	0.890189	0.850866	0.039322	0.004251	0.033072	0.930616	0.903272	0.027344
2	TF-IDF	Logistic Regression - [solver: lbfgs]	0.890189	0.850866	0.039322	0.004251	0.033072	0.930633	0.903188	0.027445
3	TF-IDF	Logistic Regression - [solver: newton-cg]	0.890189	0.850866	0.039322	0.004251	0.033072	0.930635	0.903188	0.027447
4	TF-IDF	Random Forest	0.900802	0.847944	0.052858	0.003933	0.045535	0.957656	0.897262	0.060395
5	TF-IDF	SVC - [kernel: poly]	0.896578	0.847002	0.049576	0.003524	0.039243	0.905438	0.854690	0.050747
6	Bag of Words	Bernoulli	0.882933	0.845012	0.037920	0.003072	0.034061	0.937264	0.898412	0.038853
7	TF-IDF	Bernoulli	0.882933	0.845012	0.037920	0.003072	0.034061	0.937264	0.898412	0.038853
8	TF-IDF	SVC - [kernel: sigmoid]	0.881200	0.844080	0.037121	0.003237	0.037964	0.908762	0.881717	0.027045
9	TF-IDF	SVC - [kernel: linear]	0.887698	0.843118	0.044580	0.003094	0.036253	0.907317	0.871632	0.035685
10	TF-IDF - [min_df: 5]	SVC - [kernel: rbf]	0.854884	0.838226	0.016659	0.004659	0.042292	0.844671	0.823951	0.020720

...

37	Bag of Words - [min_df: 5]	KNN - [n_neighbors: 3]	0.859974	0.801104	0.058870	0.004218	0.038981	0.883034	0.812446	0.070588
38	TF-IDF	KNN - [n_neighbors: 2]	0.814702	0.774910	0.039793	0.016428	0.042470	0.899022	0.835368	0.063654
39	TF-IDF - [min_df: 5]	KNN - [n_neighbors: 2]	0.776907	0.762260	0.014648	0.018349	0.057092	0.833784	0.803110	0.030673
40	Bag of Words - [min_df: 5]	KNN - [n_neighbors: 2]	0.816762	0.761279	0.055483	0.009292	0.049357	0.866945	0.791195	0.075750
41	Bag of Words	KNN - [n_neighbors: 2]	0.823909	0.746678	0.077231	0.018303	0.033918	0.926534	0.816691	0.109842
42	Bag of Words - [min_df: 5]	SVC - [kernel: rbf]	0.759691	0.729983	0.029708	0.006550	0.028520	0.902445	0.870064	0.032381
43	Bag of Words	SVC - [kernel: rbf]	0.755578	0.714382	0.041195	0.005464	0.030526	0.919260	0.874206	0.045053
44	Bag of Words - [min_df: 5]	SVC - [kernel: sigmoid]	0.696230	0.688083	0.008147	0.006875	0.033447	0.857413	0.846267	0.011146
45	Bag of Words	SVC - [kernel: sigmoid]	0.689516	0.672501	0.017015	0.003921	0.022026	0.878113	0.847238	0.030875
46	Bag of Words	SVC - [kernel: poly]	0.672623	0.660832	0.011791	0.001305	0.009961	0.892707	0.822122	0.070585
47	Bag of Words - [min_df: 5]	SVC - [kernel: poly]	0.668400	0.659861	0.008539	0.001269	0.013632	0.850246	0.808610	0.041636

- **Nhận xét:**

- Các thuật toán như SVM, Logistic Regression và Random Forest hoạt động khá tốt trên dữ liệu này.
- Xem ra, các model này hoạt động tốt hơn trên vectorizer TF-IDF mà ta không lọc bớt các emoji. Tuy nhiên, ta khó có thể giải thích cho việc này. \Rightarrow Ta sẽ dùng vectorizer này làm input cho toàn bộ các model về sau.
- Accuracy **train/test_acc_acc** giữa training data và test data luôn đạt trên 85% ở 5 model đầu tiên, sự chênh lệch giữa train và test **diff_acc/roc_auc** thấp hơn 5%, ở ROC-AUC **train/test_roc_auc** cũng đạt trên 85%, chứng tỏ các emoji đang làm khá tốt vai trò của chúng để phân lớp và không có class nào nổi trội hơn class nào.
- Nhìn vào phương sai **train/test_acc_std**, ta thấy được sự ổn định của model ở qua các cross-validation khác nhau. Tức không có một tập train-validation nào mà nó lại phù hợp quá mức với model.



- Nhận xét:

- Biểu đồ trên thể hiện các độ đo **train_acc**, **test_acc**, **train_roc_auc** và **test_roc_auc** được sắp xếp giảm dần theo **test_acc**.
 - Nhìn vào đây, ta thấy sự khác biệt giữa các model là không quá lớn.
- Bây giờ ta sẽ chọn ra một vài model mà nhóm cảm thấy là ổn nhất để tiến hành Tuning Hyperparameter bằng Grid-Search.
- Ta sẽ thiết lập một vài hyperparams cho các model sau đó tiến hàng training lại trên cross-validation vừa rồi.

```

1 lst_tunning_models = [
2     ('Logistic Regression - [solver: lbfsgs]', LogisticRegression(solver='lbfsgs'), {
3         'C': [0.001, 0.01, 0.1, 1.0, 10, 100],
4         'penalty': ['none', 'l1', 'l2', 'elasticnet']
5     }),
6     ('SVC - [kernel: rbf]', SVC(kernel='rbf', probability=True, random_state=42), {
7         'gamma': [0.0001, 0.001, 0.01, 1.0, 10],
8         'C': [0.1, 0.5, 1.0, 10, 25, 50, 75, 100]
9     }),
10    ('Random Forest', RandomForestClassifier(random_state=42), {
11        'bootstrap': [False, True],
12        'max_features': ['auto', 'sqrt', 'log2', 5, 10, 15, 20],
13        'n_estimators': [100, 200, 300]
14    })
15 ]

```

- Tiến hành train và lựa ra các hyperparameter set cho từng model.

```

1 lst_tunning_models = Model.trainTunningModel(lst_tunning_models,
2                                         vectorizers[1][1][1], y_train, cv)
3
4 lst_tunning_models

Model Logistic Regression - [solver: lbfgs] has been tunned in 2.55 seconds
Model SVC - [kernel: rbf] has been tunned in 23.04 seconds
Model Random Forest has been tunned in 140.28 seconds

[('Logistic Regression - [solver: lbfgs]', LogisticRegression()),
 ('SVC - [kernel: rbf]', SVC(gamma=1.0, probability=True, random_state=42)),
 ('Random Forest',
  RandomForestClassifier(bootstrap=False, max_features=5, random_state=42))]

```

- Train lại từng model với hyperparameter set trả về trên toàn bộ training data (không chia validation nữa).

```
1 tunning_models = Model.train(lst_tunning_models, [vectorizers[1]], y_train, cv)
```

TF-IDF:

```

Model Logistic Regression - [solver: lbfgs] has been trained in 0.16 seconds
Model SVC - [kernel: rbf] has been trained in 0.72 seconds
Model Random Forest has been trained in 2.32 seconds

```

```
1 tunning_models
```

	vectorizer	model	train_acc	test_acc	diff_acc	train_acc_std	test_acc_std	train_roc_auc	test_roc_auc	diff_roc_auc	tr
0	TF-IDF	SVC - [kernel: rbf]	0.893546	0.851875	0.041671	0.003487	0.038218	0.908397	0.866576	0.041821	
1	TF-IDF	Logistic Regression - [solver: lbfgs]	0.890189	0.850866	0.039322	0.004251	0.033072	0.930633	0.903188	0.027445	
2	TF-IDF	Random Forest	0.900802	0.850857	0.049945	0.003933	0.045930	0.963234	0.891460	0.071774	

- **Nhận xét:** SVC vẫn là model tốt nhất với bộ tham số mặc định với kernel **rbf**. Tuy nhiên sự khác biệt giữa các model là không quá lớn trên **test_acc**, chỉ hơn thua nhau 1%.
- Load dữ liệu test lên để đánh giá và vẫn chọn vectorizer TF-IDF đại diện cho toàn bộ dataset emoji.

```
1 X_test, y_test = Model.loadData("./data/emoji_data/test")
```

```

1 tfidf_test_emojis = vectorizers[1][1][0].transform(X_test['emoji_decode'])
2 best_models = Model.evaluation(lst_tunning_models, vectorizers[1][1][1],
3                                 y_train, tfidf_test_emojis, y_test)
4
5 best_models

```

	model	train_acc	test_acc	train_roc_auc	test_roc_auc
0	Logistic Regression - [solver: lbfgs]	0.887914	0.824903	0.865063	0.790367
1	SVC - [kernel: rbf]	0.891813	0.821012	0.869359	0.791302
2	Random Forest	0.899610	0.793774	0.889707	0.771456

- **Nhận xét:**

- Hai model Logistic và SVC có hiệu suất có thể chấp nhận được khi độ lệch giữa **train_acc** và **test_acc** dưới 10%.
 - Tuy nhiên, lúc này ta có thể thấy test data của ta có sự nỗi trội hơn của một trong hai class positive và negative so với class còn lại dựa vào **test_roc_auc**. Điều này có thể do hai lí do, test data của ta thực sự bị lệch hoặc model của ta bị overfitting, nhưng khả năng cao là test data bị lệch vì **test_roc_auc** có hiệu suất vẫn chấp nhận được.
 - Random Forest lúc này bị overfitting, chênh lệch giữa **train_acc** và **test_acc** đã chạm ngưỡng 10%.
- Tiến hành kiểm tra các hyper-params sau khi tuning trên 3 model trên.

```

1 lst_tunning_models

[('Logistic Regression - [solver: lbfgs]', LogisticRegression()),
 ('SVC - [kernel: rbf]', SVC(gamma=1.0, probability=True, random_state=42)),
 ('Random Forest',
 RandomForestClassifier(bootstrap=False, max_features=5, random_state=42))]

```

- **Nhận xét:**

- Ở đây, chỉ có hai model là Random Forest và SVC là có hyper-params thay đổi sau quá trình tuning.
 - Model Logistic vẫn giữ nguyên các hyper-params mặc định của sklearn.
- Bây giờ, ta sẽ lưu các model bên trên lại cùng với vectorizer vào cùng một object do ta định nghĩa là **SentimentModel**. Mục đích của chúng ta là để cho ta dễ dàng tái sử dụng chúng về sau và cũng như lưu trữ thành cùng một file ***.pickle** duy nhất.

```

1 logistic_model = Model.SentimentModel(lst_tunning_models[0][1], vectorizers[1], y_train)
2 svc_model = Model.SentimentModel(lst_tunning_models[1][1], vectorizers[1], y_train)
3 rforest_model = Model.SentimentModel(lst_tunning_models[2][1], vectorizers[1], y_train)
4
5 print(logistic_model.info())
6 print(svc_model.info())
7 print(rforest_model.info())

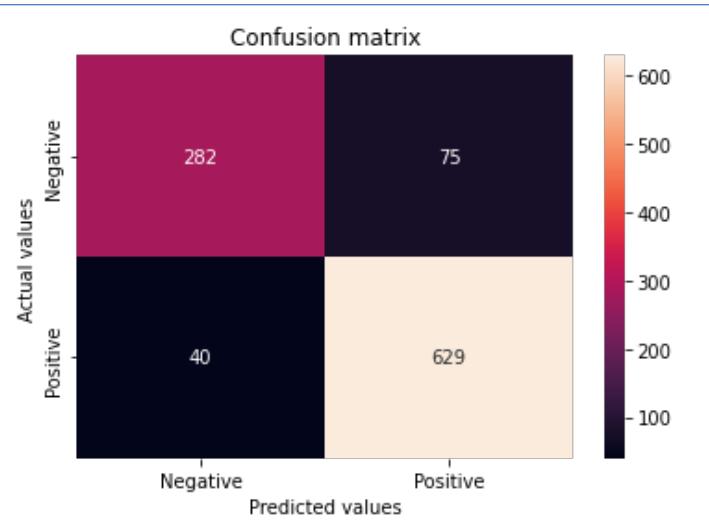
LogisticRegression()
None
SVC(gamma=1.0, probability=True, random_state=42)
None
RandomForestClassifier(bootstrap=False, max_features=5, random_state=42)
None

```

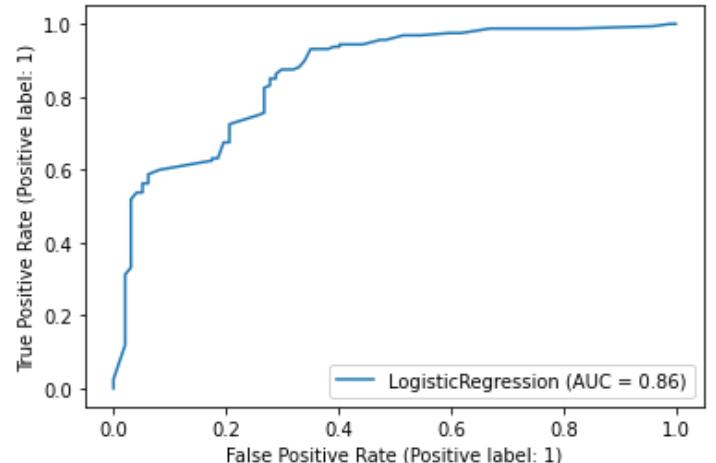
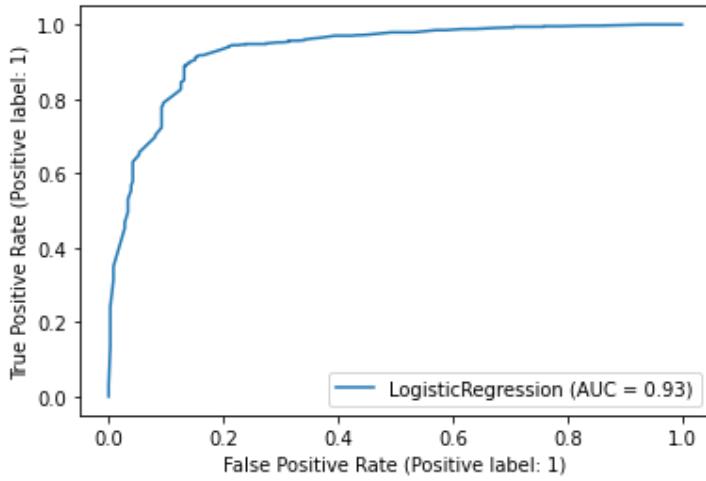
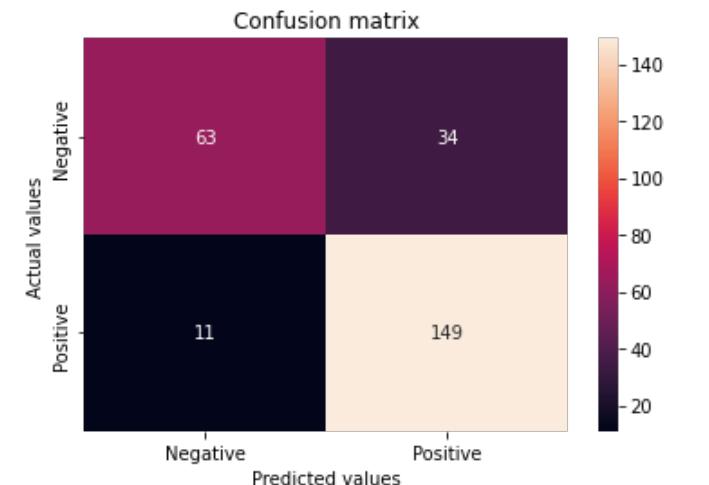
- Ở phần này, ta tiến hành đánh giá và trực quan kết quả đánh giá trên toàn bộ training data và test data bằng hai độ đo là accuracy và ROC-AUC.

- Logistic model

	precision	recall	f1-score	support
Negative	0.88	0.79	0.83	357
Positive	0.89	0.94	0.92	669
accuracy			0.89	1026
macro avg	0.88	0.87	0.87	1026
weighted avg	0.89	0.89	0.89	1026



	precision	recall	f1-score	support
Negative	0.85	0.65	0.74	97
Positive	0.81	0.93	0.87	160
accuracy			0.82	257
macro avg	0.83	0.79	0.80	257
weighted avg	0.83	0.82	0.82	257



	input	output_proba	output_class
0	heart_eyes	(0.0690459925648299, 0.9309540074351701)	1
1	hearts hearts hearts kissing_heart kissing_hear...	(0.0639276610567524, 0.9360723389432476)	1
2	relieved	(0.7222422050450168, 0.2777577949549832)	0
3	sob sob sob persevere persevere persevere pers...	(0.6086211652496625, 0.39137883475033747)	0
4	heart heart	(0.03977199088274086, 0.9602280091172591)	1
...
1021	gift_heart	(0.3500220073124234, 0.6499779926875766)	1
1022	heart	(0.03977199088274086, 0.9602280091172591)	1
1023	hugs wink	(0.3321253514338075, 0.6678746485661925)	1
1024	blush blush blush blush blush blush blus...	(0.13724536542157273, 0.8627546345784273)	1
1025	wink	(0.3712072811835553, 0.6287927188164447)	1

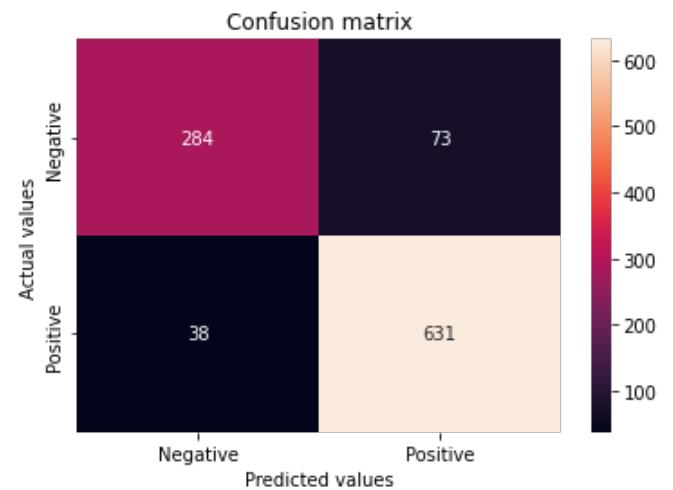
1026 rows × 3 columns

	input	output_proba	output_class
0	wink	(0.3712072811835553, 0.6287927188164447)	1
1	heart	(0.03977199088274086, 0.9602280091172591)	1
2	rofl rofl	(0.3070906240706651, 0.6929093759293349)	1
3	rofl rofl rofl rofl rofl rofl rofl rofl r...	(0.1682098403308303, 0.8317901596691697)	1
4	heart_eyes heart_eyes	(0.0690459925648299, 0.9309540074351701)	1
...
252	shit	(0.5461358590363463, 0.4538641409636537)	0
253	joy	(0.2386855657399103, 0.7613144342600897)	1
254	disappointed disappointed disappointed disappo...	(0.7608262221340357, 0.23917377786596428)	0
255	cry	(0.6713100653363226, 0.3286899346636773)	0
256	kissing_heart kissing_heart kissing_heart kiss...	(0.09632266198438322, 0.9036773380156168)	1

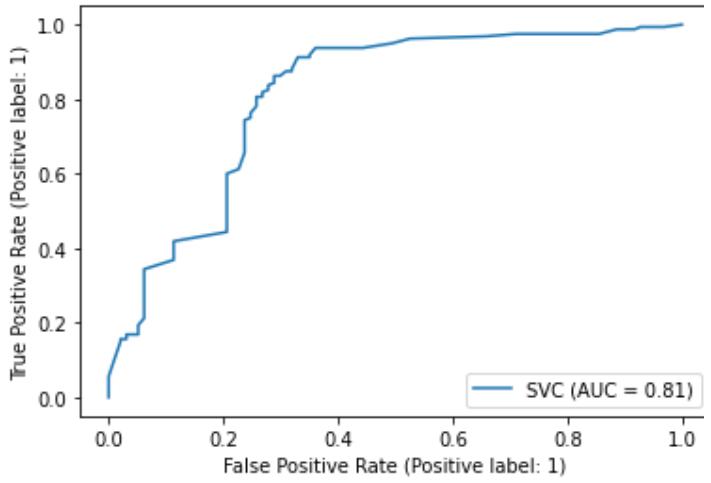
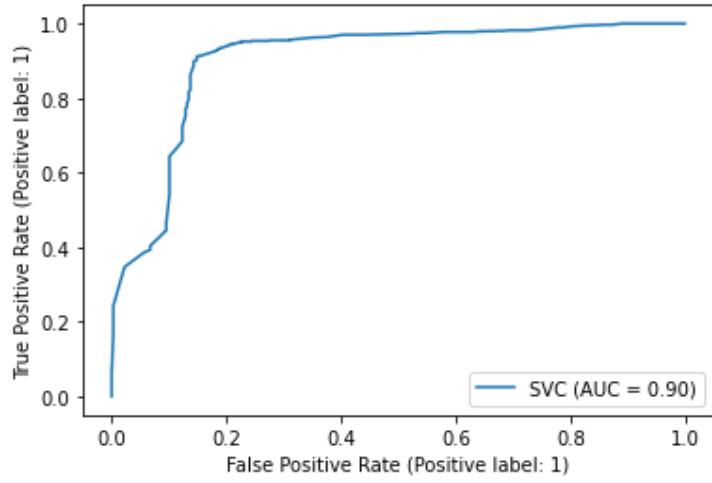
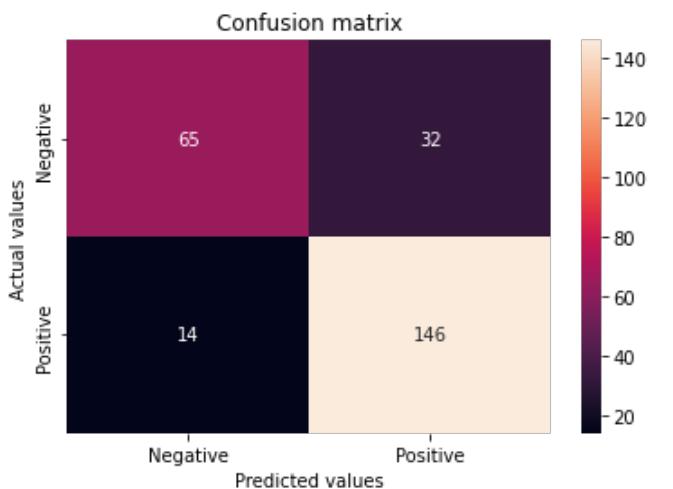
257 rows × 3 columns

- SVC model**

	precision	recall	f1-score	support
Negative	0.88	0.80	0.84	357
Positive	0.90	0.94	0.92	669
accuracy			0.89	1026
macro avg	0.89	0.87	0.88	1026
weighted avg	0.89	0.89	0.89	1026



	precision	recall	f1-score	support
Negative	0.82	0.67	0.74	97
Positive	0.82	0.91	0.86	160
accuracy			0.82	257
macro avg	0.82	0.79	0.80	257
weighted avg	0.82	0.82	0.82	257



		input	output_proba	output_class
0		heart_eyes	(0.10266231365321639, 0.8973376863467836)	1
1	hearts hearts hearts kissing_heart kissing_he...	(0.10232899031493847, 0.8976710096850615)		1
2		relieved	(0.867374421854991, 0.13262557814500886)	0
3	sob sob sob persevere persevere persevere pers...	(0.7574393894608992, 0.24256061053910113)		0
4		heart heart	(0.09273168262614771, 0.9072683173738522)	1
...	
1021		gift_heart	(0.10270967458866774, 0.8972903254113321)	1
1022		heart	(0.09273168262614771, 0.9072683173738522)	1
1023		hugs wink	(0.10272172235742406, 0.897278277642576)	1
1024	blush blush blush blush blush blush blus...	(0.10269418172196916, 0.8973058182780309)		1
1025		wink	(0.14995291012493978, 0.8500470898750603)	1

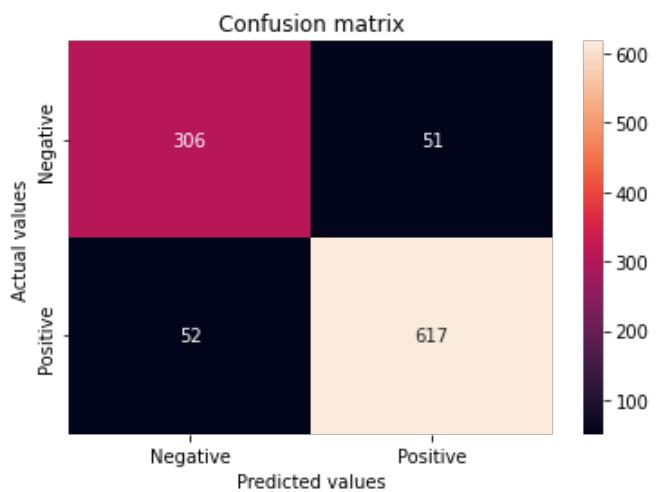
1026 rows × 3 columns

		input	output_proba	output_class
0		wink	(0.14995291012493978, 0.8500470898750603)	1
1		heart	(0.09273168262614771, 0.9072683173738522)	1
2		rofl rofl	(0.10263782191175236, 0.8973621780882477)	1
3	rofl rofl rofl rofl rofl rofl rofl rofl r...	(0.09508878763041478, 0.9049112123695854)		1
4		heart_eyes heart_eyes	(0.10266231365321639, 0.8973376863467836)	1
...	
252		shit	(0.7657189508922359, 0.23428104910776407)	0
253		joy	(0.10265509139923557, 0.8973449086007645)	1
254	disappointed disappointed disappointed disappo...	(0.8672722047277136, 0.13272779527228637)		0
255		cry	(0.8674118652597433, 0.13258813474025677)	0
256	kissing_heart kissing_heart kissing_heart kiss...	(0.10261427602171362, 0.8973857239782862)		1

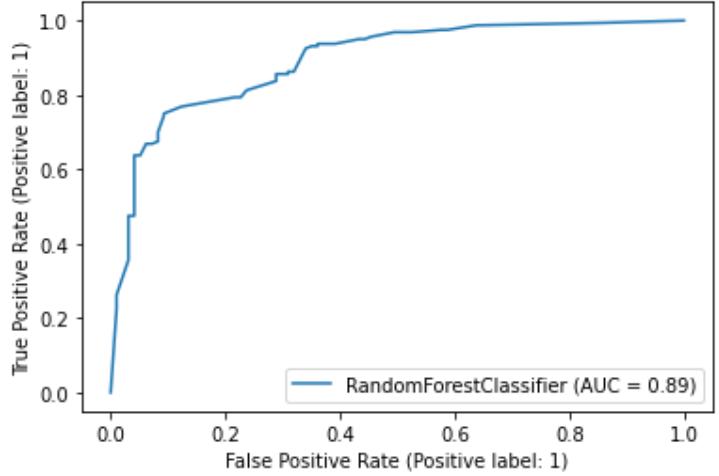
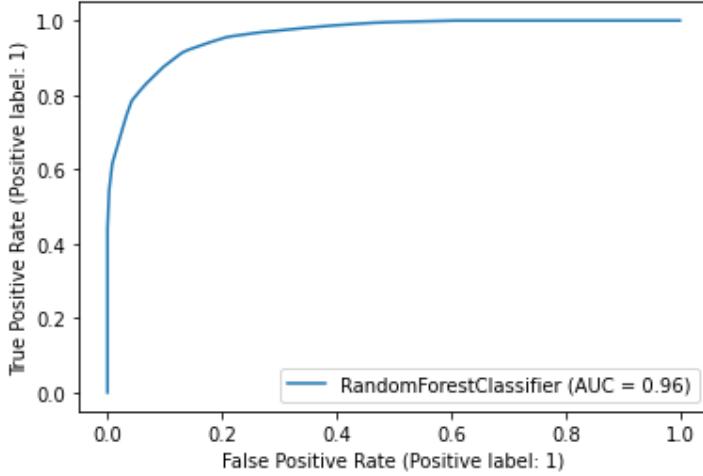
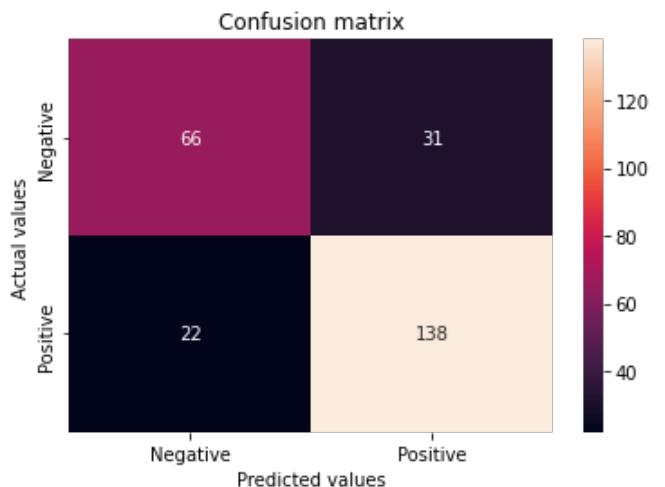
257 rows × 3 columns

- Random forest model

	precision	recall	f1-score	support
Negative	0.85	0.86	0.86	357
Positive	0.92	0.92	0.92	669
accuracy			0.90	1026
macro avg	0.89	0.89	0.89	1026
weighted avg	0.90	0.90	0.90	1026



	precision	recall	f1-score	support
Negative	0.75	0.68	0.71	97
Positive	0.82	0.86	0.84	160
accuracy			0.79	257
macro avg	0.78	0.77	0.78	257
weighted avg	0.79	0.79	0.79	257



	input	output_proba	output_class
0	heart_eyes	(0.03921568627450975, 0.9607843137254879)	1
1	hearts hearts hearts kissing_heart kissing_hear...	(0.0, 1.0)	1
2	relieved	(0.8333333333333333, 0.166666666666666646)	0
3	sob sob sob persevere persevere persevere pers...	(0.0, 1.0)	1
4	heart heart	(0.014285714285714256, 0.9857142857142847)	1
...
1021	gift_heart	(0.0, 1.0)	1
1022	heart	(0.014285714285714256, 0.9857142857142847)	1
1023	hugs wink	(0.0, 1.0)	1
1024	blush blush blush blush blush blush blus...	(0.09523809523809508, 0.904761904761903)	1
1025	wink	(0.5, 0.5)	0

1026 rows × 3 columns

	input	output_proba	output_class
0	wink	(0.5, 0.5)	0
1	heart	(0.014285714285714256, 0.9857142857142847)	1
2	rofl rofl	(0.3125, 0.6875)	1
3	rofl rofl rofl rofl rofl rofl rofl rofl r...	(0.04983552631578947, 0.9501644736842105)	1
4	heart_eyes heart_eyes	(0.03921568627450975, 0.9607843137254879)	1
...
252	shit	(1.0, 0.0)	0
253	joy	(0.2631578947368424, 0.7368421052631585)	1
254	disappointed disappointed disappointed disappo...	(0.9000000000000008, 0.0999999999999981)	0
255	cry	(0.7777777777777771, 0.2222222222222177)	0
256	kissing_heart kissing_heart kissing_heart kiss...	(0.0, 1.0)	1

257 rows × 3 columns

- Nhận xét:

- Model Random forest bị hiện tượng overfitting, hai model còn lại có xu hướng nhận diện sai các mẫu theo đúng là negative nhưng lại nhầm sang positive.
 - Tuy nhiên, nhìn chung thì hai model logistic và svm vẫn tốt, accuracy trên 80% - đây cũng là một con số khá ổn cho một tập dữ liệu dạng text như vậy.
 - Tiếp theo, nhờ vào trực quan ta cũng lí giải được lí do vì sao ROC-AUC lại thấp vì số true negative sample là khoảng 100 trong khi số true positive sample là 150.
- Ta tiến hành lưu các model này lại kèm theo vectorizer TF-IDF dưới dạng file ***.pickle**.

- Nhìn chung, ta thấy model Logistic hoạt động tốt nhất, tuy nhiên ta chưa biết liệu khi kết hợp với **Comment sentiment model** thì hiệu quả kết hợp giữa chúng sẽ như thế nào, nên ta sẽ lưu toàn bộ lại.

```

1 Model.saveByPickle(svc_model, "./models/svc_model_emojis.pickle")
2 Model.saveByPickle(logistic_model, "./models/logistic_model_emojis.pickle")
3 Model.saveByPickle(rdforest_model, "./models/rdforest_model_emojis.pickle")

```

```

<modules.model.SentimentModel object at 0x7ff6cd8adac0> has been saved at ./models/svc_model_
emojis.pickle.
<modules.model.SentimentModel object at 0x7ff6cd8adbb0> has been saved at ./models/logistic_m
odel_emojis.pickle.
<modules.model.SentimentModel object at 0x7ff7a53d8940> has been saved at ./models/rdforest_m
odel_emojis.pickle.

```

3.2.2 Công việc 2.2 (File 07.model.ipynb)

- Phần này, ta sẽ đào tạo một **Comment sentiment model**.
- Chiến lược của ta sẽ như sau:

- **Bước 1:** Tiến hành thay thế các từ trong comment bằng các từ ghép phổ biến phát sinh bởi N-Grams.
- **Bước 2:** Tiếp theo, định nghĩa một vài vectorizer bằng **sklearn**.
- **Bước 3:** Liệt kê các classifier model.
- **Bước 4:** Với từng vectorizer đã liệt kê, sử dụng từng model classifier để đào tạo cross-validation với input là vectorizer tương ứng.
- **Bước 5:** Lựa chọn các model và vectorizer phù hợp với dataset comment.
- **Bước 6:** Thực hiện tuning hyperparams cho các model được chọn ở bước 5 bằng cross-validation.
- **Bước 7:** Sau khi tuning và nhận được các hyperparams tốt nhất, tiến hành training lại nhưng không cross-validation.
- **Bước 8:** Đánh giá bằng accuracy và ROC-AUC trên test data.
- **Bước 9:** Lưu lại các model này thành file **.pickle**.

- Load training data.

	raw_comment	normalize_comment	emoji_decode		label
0	form k đẹp lắm	form không đẹp lắm		0	0
1	Áo Rộng thật sự\nGi nhanh\nChất lượ...	áo rộng thật sự nhanh chất lượng v...		1	1
2	Màu túi hơi tối do với ảnh chụp	màu túi hơi tối do ảnh chụp		2	0
3	Chất liệu vải k ổn lắm	chất liệu vải không ổn lắm		3	0
4	Mã "màu đen" hiện lên áo màu đen, còn ...	mã màu đen hiện lên áo màu đen còn mã...		4	0
...
11359	Khi bán combo ghi 3 đôi, nhưng nhận chỉ ...	bán combo ghi đôi nhưng nhận chỉ đôi ta...		11359	0
11360	Áo đẹp, from chuẩn. Sẽ ủng hộ tiếpạ !	áo đẹp from chuẩn sẽ ủng hộ tiếp		11360	1
11361	Đặt màu trắng kem thì giao màu xanh đen...	đặt màu trắng kem giao màu xanh đen giao...	expressionless	11361	0
11362	Đồ chơi tí hon.\nShop nên dừng bán sp ...	đồ chơi tí hon nên dừng bán sản phẩm...		11362	0
11363	Son đẹp lắm nha mọi ng, đáng đồng tiề...	son đẹp lắm mọi đáng đồng tiền bát m...		11363	1
11364 rows × 3 columns					

- Ở project 2, ta đã đề cập đến N-Grams với tổ hợp 2, 3 và 4 từ ghép. Nay giờ ta sẽ tạo một DataFrame để lưu các từ ghép này, cột **freq_doc** là số comment chứa cụm từ đó, cột **freq** là tần số xuất hiện của từ ghép đó trên toàn bộ dataset comment. Tất cả được sắp xếp giảm dần theo **freq_doc**.

```

1 ngram_words = Model.getDashWords(X_train['normalize_comment'], [2, 3, 4])
2
3 ngram_words

```

	freq_doc	freq
giao hàng	1914	1914
sản phẩm	1803	1803
chất lượng	1377	1377
hàng nhanh	1082	1082
giao hàng nhanh	1027	1027
...
sáp không khô môi	1	1
không khô môi lưu	1	1
khô môi lưu màu	1	1
môi lưu màu cũng	1	1
lưu màu cũng lâu	1	1

293542 rows × 2 columns

```

1 print(len(ngram_words[ngram_words['freq_doc'] >= 10]))
2 print(len(ngram_words[ngram_words['freq_doc'] >= 20]))

```

3755
1540

- **Nhận xét:**

- Có tổng cộng 3755 từ ghép mà có hơn 10 comment chứa sự xuất hiện của chúng.
- Có tổng cộng 1540 từ ghép mà có hơn 20 comment chứa sự xuất hiện của chúng.

- Nay giờ, ta tạo ra hai feature là **dash_comment** và **dash_comment_min_20**, hai cột này lần lượt chứa các comment mà các khoảng trắng trong từ các từ ghép được thay thế bằng kí tự “_”.
- Ở đây **dash_comment** sẽ thay toàn bộ các từ ghép phát sinh bởi N-Grams.
- **dash_comment_min_20** chỉ thay thế các từ ghép mà phải có ít nhất 20 comment chứa chúng. Nhưng bước này ta đã thực hiện thừa vì các vectorizing của **sklearn** hỗ trợ ta việc này.

```

1 X_train['dash_comment'] = Model.replaceInNGrams(X_train['normalize_comment'], [2, 3, 4],
2                                     ngram_words, 'freq_doc')
3 X_train['dash_comment_min_20'] = Model.replaceInNGrams(X_train['normalize_comment'], [2, 3, 4],
4                                     ngram_words, 'freq_doc', 20)
5
6 X_train

```

	raw_comment	normalize_comment	emoji_decode	dash_comment	dash_comment_min_20
0	form k đẹp lắm	form không đẹp lắm		form_không đẹp_lắm	form_không đẹp_lắm
1	Áo Rộng thật sự\nGi nhanh\nChất lướt...	áo rộng thật sự nhanh chất lượng v...		áo_rộng thật_sự nhanh_chất_lượng v...	áo_rộng thật_sự nhanh chất_lượng v...
2	Màu túi hơi tối do với ảnh chụp	màu túi hơi tối do ảnh chụp		màu túi_hơi tối_do ảnh_chụp	màu túi_hơi tối do ảnh_chụp
3	Chất liệu vải k ổn lắm	chất liệu vải không ổn lắm		chất_liệu vải_không ổn_lắm	chất_liệu vải_không ổn_lắm
4	Mã "màu đen" hiện lên áo màu đen, còn ...	mã màu đen hiện lên áo màu đen còn mã...		mã_màu_den hiện_lên_áo_màu_den còn_mã...	mã_màu_den hiện_lên_áo_màu_den_còn mã...
...
11359	Khi bán combo ghi 3 đôi, nhưng nhận chỉ ...	bán combo ghi đôi nhưng nhận chỉ đôi ta...		bán_combo ghi_đôi nhưng_nhận chỉ đôi_ta...	bán_combo ghi_đôi nhưng_nhận chỉ đôi_ta...
11360	Áo đẹp, from chuẩn. Sẽ ủng hộ tiếp ạ !	áo đẹp from chuẩn sẽ ủng hộ tiếp		áo_đẹp from_chuẩn sẽ_ủng_hỗ tiếp	áo_đẹp from_chuẩn sẽ_ủng_hỗ tiếp
11361	Đặt màu trắng kem thì giao màu xanh đen giao...	đặt màu trắng kem giao màu xanh đen giao...	expressionless	đặt_màu_trắng_kem giao_màu_xanh đen_giao...	đặt_màu_trắng_kem giao_màu_xanh đen_giao...
11362	Đồ chơi tí hon.\nShop nên dừng bán sp ...	đồ chơi tí hon nên dừng bán sản phẩ...		đồ_chơi_tí_hon_nên_dừng_bán sản_phẩ...	đồ_chơi_tí_hon_nên_dừng_bán sản_phẩ...
11363	Son đẹp lắm nha mọi ng, đáng đồng tiề...	son đẹp lắm mọi_đáng đồng tiền bát m...		son đẹp_lắm mọi_đáng đồng tiền_bát m...	son đẹp_lắm mọi_đáng đồng tiền_bát m...

11364 rows × 5 columns

- Dưới đây ta lần lượt phát sinh các vectorizer bằng cả Bag of Words và TF-IDF với các **min_df** khác nhau là 5, 10 và 20.

```

1 vectorizers = [
2     ("Bag of Words", Model.vectorizer(X_train['dash_comment'], 'bow')),
3     ("TF-IDF", Model.vectorizer(X_train['dash_comment'], 'tfidf')),
4     ("Bag of Words- [min_df: 5]", Model.vectorizer(X_train['dash_comment'], 'bow', 5)),
5     ("TF-IDF- [min_df: 5]", Model.vectorizer(X_train['dash_comment'], 'tfidf', 5)),
6     ("Bag of Words - [min_df: 10]", Model.vectorizer(X_train['dash_comment'], 'bow', 10)),
7     ("TF-IDF - [min_df: 10]", Model.vectorizer(X_train['dash_comment'], 'tfidf', 10)),
8     ("Bag of Words - [min_df: 20]", Model.vectorizer(X_train['dash_comment'], 'bow', 20)),
9     ("TF-IDF - [min_df: 20]", Model.vectorizer(X_train['dash_comment'], 'tfidf', 20))
10 ]
11
12 ix = 0
13 print(len(vectorizers[ix][1][0].get_feature_names()))
14 print(vectorizers[ix][1][0].get_feature_names()[:100])
15 pd.DataFrame(data=vectorizers[ix][1][0].toarray(), columns=vectorizers[ix][1][0].get_feature_names())
20010
['a_a', 'a_av', 'a_bê', 'a.Cho', 'a_cùng', 'a_e', 'a_hong', 'a_hời', 'a_rất', 'a_xâ', 'ac_clone', 'ace', 'ad', 'ad_bo
c', 'ad_gửi', 'ad_vào', 'ah', 'ah_ái', 'ah_nhen', 'ai', 'ai_chân', 'ai_có', 'ai_cũng', 'ai_cá', 'ai_cần', 'ai_dám',
'ai_dè', 'ai_eo', 'ai_không', 'ai_luôn', 'ai_làm', 'ai_mua', 'ai_muốn', 'ai_má', 'ai_mắt', 'ai_mặc', 'ai_nghĩ', 'ai_n
gờ', 'ai_nhận', 'ai_nhắn', 'ai_review', 'ai_ranh', 'ai_rõng', 'ai_tay', 'ai_thích', 'ai_trắng', 'ai_về', 'ai_dời', 'a
m', 'amazing', 'an', 'an_lành', 'an_toàn', 'and', 'and_tacos', 'and_think', 'android_gôm', 'anh', 'anh_chú', 'anh_cut
e', 'anh_có', 'anh_de', 'anh_em', 'anh_giữ', 'anh_không', 'anh_muốn', 'anh_nên', 'anh_nói', 'anh_ship', 'anh_siêu',
'anh_dáng', 'anh_oi', 'anh_ấy', 'any_nên', 'ao', 'ao_cũng', 'ao_kha', 'ao_khác', 'ao_nhu', 'ao_thí', 'ao_thun', 'ao_v
ua', 'auto', 'auto_hay', 'b_có', 'b_nào', 'b_nên', 'ba', 'ba_cai', 'ba_chăm', 'ba_con', 'ba_cái', 'ba_lô', 'ba_tia',
'ba_dời', 'bai', 'bai_giân', 'bam', 'ban', 'ban_cái']

```

a_a	a_av	a_bé	a_cho	a_cùng	a_e	a_hồng	a_hời	a_rất	a_xã	...	ứng_học	ức_chế	ức_x	ức_óc_ức_of	ứng_dụng_trên_android	ứng_luôn	ù_mưa
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
...
11359	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
11360	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
11361	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
11362	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
11363	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0

11364 rows × 20010 columns

- Nhận xét:

- Với vectorizer đầu tiên, ta thiết lập **min_df=1** mặc định của **sklearn** thì ta có tổng cộng 20,000 từ (kể cả từ ghép).
- Số lượng này là quá lớn, sẽ khiến việc train model sẽ lâu.

```

1 ix = 3
2 print(len(vectorizers[ix][1][0].get_feature_names()))
3 print(vectorizers[ix][1][0].get_feature_names()[:100])
4 pd.DataFrame(data=vectorizers[ix][1][1].toarray(), columns=vectorizers[ix][1][0].get_feature_names())

```

2742

ah	ai	ai_có	ai_cùng	ai_dè	ai_mua	ai_ngờ	anh	anh_em	ao	...	ổn_so	ổn_trong	ổn_áp	ổn_dep	ở_ngoài	ở_nhà	ở_trong	ở_trên	ở_dày	ứng
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
...	
11359	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
11360	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.42
11361	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
11362	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
11363	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00

11364 rows × 2742 columns

- Nhận xét:

- Với **min_df=5**, không gian nhỏ hơn từ 20,000 xuống còn hơn 2,700 từ.

- Số lượng này ổn, thích hợp để đào tạo hơn đồng thời cho ta thấy trong hơn 11,000 comment, từ rác phát sinh quá nhiều bởi chính dữ liệu và N-Grams.

1	ix = 5																																																																																																																																																																																																																																																
2	print(len(vectorizers[ix][1][0].get_feature_names()))																																																																																																																																																																																																																																																
3	print(vectorizers[ix][1][0].get_feature_names()[:100])																																																																																																																																																																																																																																																
4	pd.DataFrame(data=vectorizers[ix][1][1].toarray(), columns=vectorizers[ix][1][0].get_feature_names())																																																																																																																																																																																																																																																
1454																																																																																																																																																																																																																																																	
['ah', 'ai', 'ai_ngờ', 'anh', 'anh_em', 'ao', 'ban_dầu', 'bao', 'bao_giờ', 'bao_nhiêu', 'bay_màu', 'biết', 'biết_là', 'bung_chỉ', 'bung_hết', 'buồn_bán', 'buồn', 'buộc_tóc', 'bán', 'bán_hàng', 'bán_dat', 'báo', 'báo_giao', 'bát_gạo', 'bé', 'bé_hơn', 'bên', 'bên_ngoài', 'bên_trong', 'bình_luận', 'bình_thường', 'bó_tay', 'bóc_ra', 'bóng', 'bóp', 'bóng_tai', 'bù', 'bạn', 'bạn_nào', 'bảo', 'bảo_hành', 'bất_ngờ', 'bản', 'bận', 'bằng', 'bèn', 'bèn_không', 'bọc', 'bỏ', 'bó_ra', 'bó_tiền', 'bồ', 'bồ_dò', 'bục_chỉ', 'bung', 'bực', 'can_than', 'cao', 'cao_mặc', 'cao_năng', 'chat', 'chí_tiết', 'chim_ứng', 'chiếc', 'chiều_dài', 'cho', 'cho_cái', 'cho_có', 'cho_khách', 'cho_làm', 'cho_làm', 'cho_người', 'cho_vì', 'cho_dồi', 'chu_dáo', 'chuyên_nghiệp', 'chuyển_khoảng', 'chuẩn', 'chuẩn_hàng', 'chán', 'chán_lầm', 'chán_quá', 'chân', 'chân_vayah', 'chú_y', 'chúc', 'chúc_buồn', 'chút', 'chút_nhưng', 'chút_não', 'chăm_sóc', 'chữa', 'chua_biết', 'chua_dùng', 'chua_mặc', 'chua_thầy', 'chua_dược', 'chả', 'chả_hiểu']																																																																																																																																																																																																																																																	
<table border="1"><thead><tr><th>ah</th><th>ai</th><th>ai_ngờ</th><th>anh</th><th>anh_em</th><th>ao</th><th>ban_dầu</th><th>bao</th><th>bao_giờ</th><th>bao_nhiêu</th><th>...</th><th>ồn</th><th>ồn_lầm</th><th>ồn_nhưng</th><th>ồn_so</th><th>ồn_ap</th><th>ờ Ngoài</th><th>ờ_nhà</th><th>ờ_trong</th><th>ờ_đây</th></tr></thead><tbody><tr><td>0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>1</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>2</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>3</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.685498</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>4</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr><tr><td>11359</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>11360</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>11361</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>11362</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>11363</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr></tbody></table>		ah	ai	ai_ngờ	anh	anh_em	ao	ban_dầu	bao	bao_giờ	bao_nhiêu	...	ồn	ồn_lầm	ồn_nhưng	ồn_so	ồn_ap	ờ Ngoài	ờ_nhà	ờ_trong	ờ_đây	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.685498	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11359	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11360	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11361	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11362	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11363	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ah	ai	ai_ngờ	anh	anh_em	ao	ban_dầu	bao	bao_giờ	bao_nhiêu	...	ồn	ồn_lầm	ồn_nhưng	ồn_so	ồn_ap	ờ Ngoài	ờ_nhà	ờ_trong	ờ_đây																																																																																																																																																																																																																														
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.685498	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
...																																																																																																																																																																																																																														
11359	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
11360	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
11361	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
11362	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
11363	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
11364 rows × 1454 columns																																																																																																																																																																																																																																																	

- Nhận xét:** Với **min_df=10**, lúc này chỉ còn hơn 1,400 từ và từ ghép thỏa phô biến trên dataset comment.

1	ix = 7																																																																																																																																																																																																																																																
2	print(len(vectorizers[ix][1][0].get_feature_names()))																																																																																																																																																																																																																																																
3	print(vectorizers[ix][1][0].get_feature_names()[-100:])																																																																																																																																																																																																																																																
4	pd.DataFrame(data=vectorizers[ix][1][1].toarray(), columns=vectorizers[ix][1][0].get_feature_names())																																																																																																																																																																																																																																																
754																																																																																																																																																																																																																																																	
['xong', 'xuất_sắc', 'xám', 'xiu', 'xù_lông', 'xấu', 'xấu_không', 'xin', 'xin_xo', 'y_hình', 'yên_tâm', 'yêu_cầu', 'á_o', 'áo_chất', 'áo_có', 'áo_cũng', 'áo_hơi', 'áo_không', 'áo_mặc', 'áo_mỏng', 'áo_ok', 'áo_quá', 'áo_rất', 'áo_thun', 'áo_trắng', 'áo_vải', 'áo_xinh', 'áo_dẹp', 'ý_kiến', 'den', 'deo', 'di', 'dáng', 'dáng_mua', 'dáng_tiền', 'dánh_giá', 'dày', 'dày_là', 'dài_hỏi', 'dôi', 'dôi_tất', 'dúng', 'dúng_là', 'dúng_màu', 'dúng_mẫu', 'dúng_như', 'dún_g_size', 'đom', 'đon_hàng', 'đường_chỉ', 'đường_may', 'được', 'được_cái', 'được_giá', 'được_nhưng', 'được_tặng', 'đẩy', 'đầu', 'đầy_dù', 'đặt', 'đặt_cái', 'đặt_hàng', 'đặt_màu', 'đặt_size', 'đặt_xl', 'đặt_áo', 'đẹp', 'đẹp_chất', 'đẹp_hon', 'đẹp_luôn', 'đẹp_lấm', 'đẹp_nhưng', 'đẹp_vải', 'đến', 'đều', 'đòi', 'đồng_hồ', 'đồng_tiền', 'đôi', 'đôi_hàng', 'đôi_size', 'đôi_trả', 'đợi', 'đù_hàng', 'đừng', 'đứng_dược', 'oi', 'ung', 'ung_luôn', 'ung_lấm', 'ung_y', 'anh', 'án_h_không', 'ấy', 'òn', 'ở Ngoài', 'ở_nhà', 'ở_dây', 'ủng_hộ']																																																																																																																																																																																																																																																	
<table border="1"><thead><tr><th>ai</th><th>ba_giờ</th><th>ba_nhiêu</th><th>biết</th><th>bung_chí</th><th>buộc_tóc</th><th>bán</th><th>bán_hàng</th><th>bán_dat</th><th>báo</th><th>...</th><th>ưng_lầm</th><th>ưng_y</th><th>anh</th><th>anh_không</th><th>ấy</th><th>ồn</th><th>ờ Ngoài</th><th>ờ_nhà</th><th>...</th></tr></thead><tbody><tr><td>0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>1</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>2</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>3</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>4</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr><tr><td>11359</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>11360</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>11361</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>11362</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.539375</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>11363</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.000000</td><td>0.0</td><td>0.0</td><td>0.0</td><td>...</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr></tbody></table>		ai	ba_giờ	ba_nhiêu	biết	bung_chí	buộc_tóc	bán	bán_hàng	bán_dat	báo	...	ưng_lầm	ưng_y	anh	anh_không	ấy	ồn	ờ Ngoài	ờ_nhà	...	0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11359	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11360	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11361	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11362	0.0	0.0	0.0	0.0	0.0	0.0	0.539375	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11363	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ai	ba_giờ	ba_nhiêu	biết	bung_chí	buộc_tóc	bán	bán_hàng	bán_dat	báo	...	ưng_lầm	ưng_y	anh	anh_không	ấy	ồn	ờ Ngoài	ờ_nhà	...																																																																																																																																																																																																																														
0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
1	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
2	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
3	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
4	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
...																																																																																																																																																																																																																														
11359	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
11360	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
11361	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
11362	0.0	0.0	0.0	0.0	0.0	0.0	0.539375	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
11363	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0																																																																																																																																																																																																																														
11364 rows × 754 columns																																																																																																																																																																																																																																																	

- **Nhận xét:** Với **min_df=20**, thì chỉ còn 754, con số này thì quá nhỏ so với hơn 11,000 comment, khả năng cao vectorizer này không đủ khả năng để đại diện cho dataset comment.
- Bây giờ, ta liệt kê một vài classifier model phổ biến và dùng các vectorizer có **min_df** lần lượt là 5, 10 và 20.
- Ta tiến hành train bằng cross-validation và chia training data làm 10 phần - sau đó lưu vào biến **cv** để đảm bảo dữ liệu khi cross-validation công bằng cho toàn bộ model.

```

1 lst_models = [
2     ('Logistic Regression - [solver: lbfgs]', LogisticRegression(solver='lbfgs')),
3     ('Logistic Regression - [solver: liblinear]', LogisticRegression(solver='liblinear')),
4     ('Logistic Regression - [solver: newton-cg]', LogisticRegression(solver='newton-cg')),
5     ('KNN - [n_neighbors: 2]', KNeighborsClassifier(n_neighbors=2)),
6     ('KNN - [n_neighbors: 3]', KNeighborsClassifier(n_neighbors=3)),
7     ('SVC - [kernel: linear]', SVC(kernel='linear', random_state=42)),
8     ('SVC - [kernel: poly]', SVC(kernel='poly', random_state=42)),
9     ('SVC - [kernel: rbf]', SVC(kernel='rbf', random_state=42)),
10    ('SVC - [kernel: sigmoid]', SVC(kernel='sigmoid', random_state=42)),
11    ('Bernoulli', BernoulliNB()),
12    ('Random Forest', RandomForestClassifier(random_state=42)),
13    ('XGBoost', XGBClassifier(eval_metric='mlogloss'))
14 ]
15
16 cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

```

```

1 original_models = Model.train(lst_models, vectorizers[2:], y_train, cv)
2
3 original_models

```

Bag of Words- [min_df: 5]:

```

Model Logistic Regression - [solver: lbfgs] has been trained in 0.96 seconds
Model Logistic Regression - [solver: liblinear] has been trained in 0.39 seconds
Model Logistic Regression - [solver: newton-cg] has been trained in 1.02 seconds
Model KNN - [n_neighbors: 2] has been trained in 31.24 seconds
Model KNN - [n_neighbors: 3] has been trained in 33.71 seconds
Model SVC - [kernel: linear] has been trained in 81.84 seconds
Model SVC - [kernel: poly] has been trained in 165.82 seconds
Model SVC - [kernel: rbf] has been trained in 141.31 seconds
Model SVC - [kernel: sigmoid] has been trained in 78.21 seconds
Model Bernoulli has been trained in 0.15 seconds
Model Random Forest has been trained in 55.12 seconds
Model XGBoost has been trained in 3.89 seconds

```

TF-IDF- [min_df: 5]:

```

Model Logistic Regression - [solver: lbfgs] has been trained in 0.58 seconds
Model Logistic Regression - [solver: liblinear] has been trained in 0.27 seconds
Model Logistic Regression - [solver: newton-cg] has been trained in 0.68 seconds
Model KNN - [n_neighbors: 2] has been trained in 31.39 seconds
Model KNN - [n_neighbors: 3] has been trained in 33.91 seconds
Model SVC - [kernel: linear] has been trained in 83.71 seconds
Model SVC - [kernel: poly] has been trained in 267.66 seconds
Model SVC - [kernel: rbf] has been trained in 137.06 seconds
Model SVC - [kernel: sigmoid] has been trained in 87.28 seconds
Model Bernoulli has been trained in 0.14 seconds
Model Random Forest has been trained in 57.32 seconds
Model XGBoost has been trained in 8.20 seconds

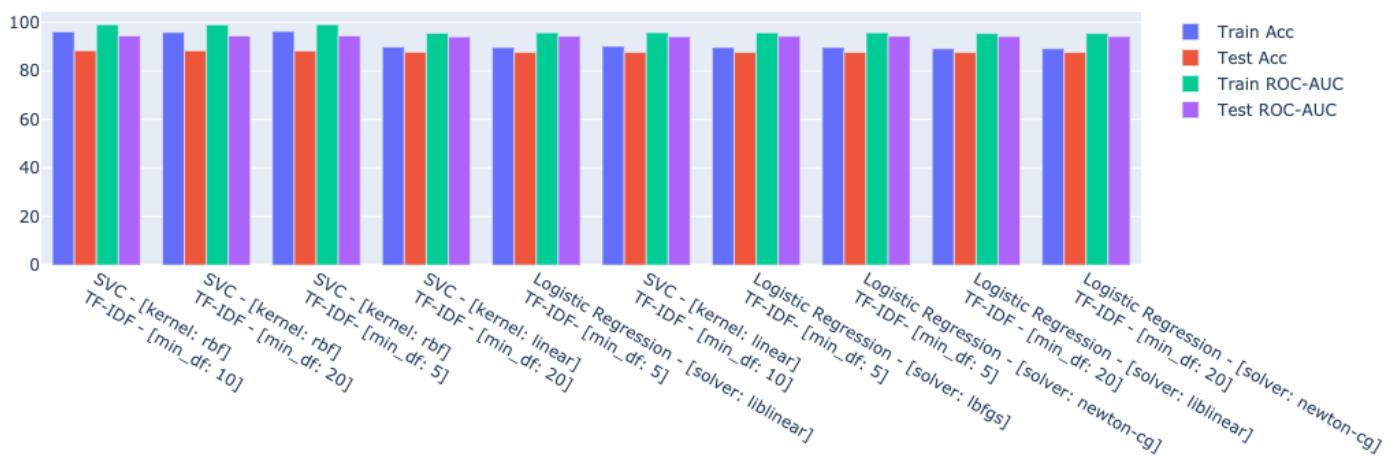
```

```
Bag of Words - [min_df: 10]:  
    Model Logistic Regression - [solver: lbfgs] has been trained in 0.80 seconds  
    Model Logistic Regression - [solver: liblinear] has been trained in 0.35 seconds  
    Model Logistic Regression - [solver: newton-cg] has been trained in 0.91 seconds  
    Model KNN - [n_neighbors: 2] has been trained in 29.91 seconds  
    Model KNN - [n_neighbors: 3] has been trained in 32.79 seconds  
    Model SVC - [kernel: linear] has been trained in 70.63 seconds  
    Model SVC - [kernel: poly] has been trained in 137.99 seconds  
    Model SVC - [kernel: rbf] has been trained in 120.32 seconds  
    Model SVC - [kernel: sigmoid] has been trained in 66.15 seconds  
    Model Bernoulli has been trained in 0.13 seconds  
    Model Random Forest has been trained in 47.82 seconds  
    Model XGBoost has been trained in 1.82 seconds  
TF-IDF - [min_df: 10]:  
    Model Logistic Regression - [solver: lbfgs] has been trained in 0.44 seconds  
    Model Logistic Regression - [solver: liblinear] has been trained in 0.24 seconds  
    Model Logistic Regression - [solver: newton-cg] has been trained in 0.57 seconds  
    Model KNN - [n_neighbors: 2] has been trained in 29.77 seconds  
    Model KNN - [n_neighbors: 3] has been trained in 32.56 seconds  
    Model SVC - [kernel: linear] has been trained in 69.90 seconds  
    Model SVC - [kernel: poly] has been trained in 218.61 seconds  
    Model SVC - [kernel: rbf] has been trained in 115.95 seconds  
    Model SVC - [kernel: sigmoid] has been trained in 74.33 seconds  
    Model Bernoulli has been trained in 0.13 seconds  
    Model Random Forest has been trained in 51.06 seconds  
    Model XGBoost has been trained in 4.79 seconds
```

```
Bag of Words - [min_df: 20]:  
    Model Logistic Regression - [solver: lbfgs] has been trained in 0.62 seconds  
    Model Logistic Regression - [solver: liblinear] has been trained in 0.31 seconds  
    Model Logistic Regression - [solver: newton-cg] has been trained in 0.82 seconds  
    Model KNN - [n_neighbors: 2] has been trained in 29.54 seconds  
    Model KNN - [n_neighbors: 3] has been trained in 32.43 seconds  
    Model SVC - [kernel: linear] has been trained in 63.66 seconds  
    Model SVC - [kernel: poly] has been trained in 113.80 seconds  
    Model SVC - [kernel: rbf] has been trained in 104.56 seconds  
    Model SVC - [kernel: sigmoid] has been trained in 57.07 seconds  
    Model Bernoulli has been trained in 0.13 seconds  
    Model Random Forest has been trained in 43.45 seconds  
    Model XGBoost has been trained in 1.47 seconds  
TF-IDF - [min_df: 20]:  
    Model Logistic Regression - [solver: lbfgs] has been trained in 0.44 seconds  
    Model Logistic Regression - [solver: liblinear] has been trained in 0.23 seconds  
    Model Logistic Regression - [solver: newton-cg] has been trained in 0.59 seconds  
    Model KNN - [n_neighbors: 2] has been trained in 29.88 seconds  
    Model KNN - [n_neighbors: 3] has been trained in 32.44 seconds  
    Model SVC - [kernel: linear] has been trained in 60.12 seconds  
    Model SVC - [kernel: poly] has been trained in 170.33 seconds  
    Model SVC - [kernel: rbf] has been trained in 101.32 seconds  
    Model SVC - [kernel: sigmoid] has been trained in 67.65 seconds  
    Model Bernoulli has been trained in 0.14 seconds  
    Model Random Forest has been trained in 50.73 seconds  
    Model XGBoost has been trained in 6.50 seconds
```

vectorizer	model	train_acc	test_acc	diff_acc	train_acc_std	test_acc_std	train_roc_auc	test_roc_auc	diff_roc_auc	train_roc_auc_std	test_roc_auc
0	TF-IDF-[min_df: 5]	SVC - [kernel: rbf]	0.960431	0.860610	0.099820	0.000862	0.006475	0.988109	0.920209	0.067900	0.000571
1	TF-IDF-[min_df: 5]	SVC - [kernel: linear]	0.909793	0.859379	0.050414	0.001363	0.007170	0.964620	0.926554	0.038066	0.000480
2	Bag of Words-[min_df: 5]	Bernoulli	0.881742	0.858763	0.022979	0.001342	0.011235	0.947202	0.925786	0.021416	0.000517
3	TF-IDF-[min_df: 5]	Bernoulli	0.881742	0.858763	0.022979	0.001342	0.011235	0.947202	0.925786	0.021416	0.000517
4	TF-IDF-[min_df: 5]	Logistic Regression - [solver: liblinear]	0.896603	0.858323	0.038280	0.001177	0.006163	0.959189	0.929278	0.029911	0.000363
...
67	TF-IDF-[min_df: 20]	KNN - [n_neighbors: 2]	0.777690	0.669394	0.108296	0.001445	0.009342	0.944053	0.739783	0.204269	0.002827
68	TF-IDF-[min_df: 10]	KNN - [n_neighbors: 3]	0.769114	0.654967	0.114147	0.074013	0.014413	0.936307	0.730836	0.205471	0.006121
69	TF-IDF-[min_df: 10]	KNN - [n_neighbors: 2]	0.725683	0.630235	0.095448	0.002135	0.013075	0.954077	0.695940	0.258137	0.001683
70	TF-IDF-[min_df: 5]	KNN - [n_neighbors: 3]	0.739878	0.617742	0.122136	0.102235	0.011546	0.950730	0.693012	0.257718	0.005090
71	TF-IDF-[min_df: 5]	KNN - [n_neighbors: 2]	0.683728	0.599964	0.083764	0.001828	0.009139	0.964292	0.663355	0.300937	0.001488

72 rows × 13 columns



- Nhận xét:

- Ta thấy 5 model đầu tiên có **train_acc** và **test_acc** cao, tuy nhiên ở model SVC sử dụng **kernel=rbf** có sự xuất hiện overfitting ở đây khi **train_acc** và **test_acc** chênh lệch chạm ngưỡng 10%.

- Ở đây, ta có thể bỏ qua **train/test roc_auc** vì ở project 2 đã cân bằng input đầu vào + **train/test acc** cao nên ta không cần xem các con số này nữa.
 - Ở các model này, vectorizer phổ biến là TF-IDF có **min_df=5**.
- Ở đây, vì nhóm đã nêu ở đầu là nhóm sẽ ưu tiên cho hai thuật toán là Logistic Regression và SVC, nhưng tuning cho SVC tốn rất nhiều thời gian (*sẽ được đề cập sau*) \Rightarrow Nên ta sẽ không tuning cho các SVC model.
- Hai dòng code mà nhóm comment bên dưới là các hyper-params được mọi người thường hay sử dụng để tuning cho model.

```

1 # 'gamma': [0.0001, 0.001, 0.01, 1.0, 10],
2 # 'C': [0.1, 0.5, 1.0, 10, 25, 50, 75, 100]
3
4 lst_tunning_models = [
5     ('Logistic Regression - [solver: liblinear]', LogisticRegression(solver='liblinear'), {
6         'C': [0.001, 0.01, 0.1, 1.0, 10, 100],
7         'penalty': ['none', 'l1', 'l2', 'elasticnet']
8     }),
9     ('Logistic Regression - [solver: newton-cg]', LogisticRegression(solver='newton-cg'), {
10        'C': [0.001, 0.01, 0.1, 1.0, 10, 100],
11        'penalty': ['none', 'l1', 'l2', 'elasticnet']
12    }),
13    ('Logistic Regression - [solver: lbfgs]', LogisticRegression(solver='lbfgs'), {
14        'C': [0.001, 0.01, 0.1, 1.0, 10, 100],
15        'penalty': ['none', 'l1', 'l2', 'elasticnet']
16    }),
17    ('SVC - [kernel: rbf]', SVC(kernel='rbf', random_state=42), {
18        'probability': [True]
19    }),
20    ('SVC - [kernel: linear]', SVC(kernel='linear', random_state=42), {
21        'probability': [True]
22    }),
23    ('SVC - [kernel: sigmoid]', SVC(kernel='sigmoid', random_state=42), {
24        'probability': [True]
25    }),
26    ('Bernoulli', BernoulliNB(), {
27        'alpha': [0.0, 0.0001, 0.001, 0.01, 0.1, 0.5, 1.0, 2.0, 3.0, 5.0, 7.0, 8.5, 10.0]
28    })
29 ]

```

```

1 lst_tunning_models = Model.trainTunningModel(lst_tunning_models, vectorizers[3][1][1], y_train, cv)
2
3 lst_tunning_models

Model Logistic Regression - [solver: liblinear] has been tunned in 2.64 seconds
Model Logistic Regression - [solver: newton-cg] has been tunned in 324.04 seconds
Model Logistic Regression - [solver: lbfgs] has been tunned in 10.82 seconds
Model SVC - [kernel: rbf] has been tunned in 332.19 seconds
Model SVC - [kernel: linear] has been tunned in 191.71 seconds
Model SVC - [kernel: sigmoid] has been tunned in 200.83 seconds
Model Bernoulli has been tunned in 0.57 seconds

[('Logistic Regression - [solver: liblinear]',
  LogisticRegression(solver='liblinear')),
 ('Logistic Regression - [solver: newton-cg]',
  LogisticRegression(solver='newton-cg')),
 ('Logistic Regression - [solver: lbfgs]',
  LogisticRegression(solver='lbfgs')),
 ('SVC - [kernel: rbf]', SVC(probability=True, random_state=42)),
 ('SVC - [kernel: linear]',
  SVC(kernel='linear', probability=True, random_state=42)),
 ('SVC - [kernel: sigmoid]',
  SVC(kernel='sigmoid', probability=True, random_state=42)),
 ('Bernoulli', BernoulliNB(alpha=0.5))]

```

- Nhận xét:

- Ta có thể thấy nếu ta thiết lập tuning cho các hyper-params là **C** và **gamma** của SVC, với GridSearch ta có 40 bộ tham số, nhìn vào model SVC kernel phía trên ta thấy thời gian cross-validation là hơn 300 giây, như vậy $300 * 40 = 12000$ giây, tức hơn 3 tiếng đồng hồ chỉ để tuning cho model này.

- Ở đây, duy nhất model **Bernoulli** là được cập nhật hyper-params sau quá trình tunnning, còn các model khác vẫn giữ nguyên các hyper-params mặc định của **sklearn**.
- Bây giờ, ta sẽ đào tạo lại các model sau khi tunnning hyper-params và đào tạo trên toàn bộ training data.

```
TF-IDF- [min_df: 5]:
    Model Logistic Regression - [solver: liblinear] has been trained in 0.25 seconds
    Model Logistic Regression - [solver: newton-cg] has been trained in 0.65 seconds
    Model Logistic Regression - [solver: lbfgs] has been trained in 0.55 seconds
    Model SVC - [kernel: rbf] has been trained in 375.36 seconds
    Model SVC - [kernel: linear] has been trained in 232.39 seconds
    Model SVC - [kernel: sigmoid] has been trained in 231.48 seconds
    Model Bernoulli has been trained in 0.14 seconds
```

vectorizer	model	train_acc	test_acc	diff_acc	train_acc_std	test_acc_std	train_roc_auc	test_roc_auc	diff_roc_auc	train_roc_auc_std	test_roc_auc_s
0	TF-IDF- [min_df: 5]	SVC - [kernel: rbf]	0.960431	0.860610	0.099820	0.000862	0.006475	0.988109	0.920209	0.067900	0.000571
1	TF-IDF- [min_df: 5]	Bernoulli	0.884166	0.860258	0.023908	0.001399	0.010917	0.949375	0.925876	0.023498	0.000498
2	TF-IDF- [min_df: 5]	SVC - [kernel: linear]	0.909793	0.859379	0.050414	0.001363	0.007170	0.964620	0.926554	0.038066	0.000480
3	TF-IDF- [min_df: 5]	Logistic Regression - [solver: liblinear]	0.896603	0.858323	0.038280	0.001177	0.006163	0.959189	0.929278	0.029911	0.000363
4	TF-IDF- [min_df: 5]	Logistic Regression - [solver: newton-cg]	0.896594	0.858323	0.038270	0.001177	0.006302	0.959186	0.929276	0.029910	0.000362
5	TF-IDF- [min_df: 5]	Logistic Regression - [solver: lbfgs]	0.896584	0.858323	0.038261	0.001181	0.006302	0.959187	0.929272	0.029915	0.000362
6	TF-IDF- [min_df: 5]	SVC - [kernel: sigmoid]	0.878867	0.857883	0.020984	0.001424	0.007155	0.942805	0.926173	0.016631	0.000780

- Nhận xét:** Sau khi model **Bernoulli** được tunnning, model này từ vị trí thứ 3 lên vị trí thứ hai dựa trên test_acc. Tuy nhiên cải thiện là không quá lớn chỉ 1%.
- Bây giờ, ta sẽ load dữ liệu test data lên để tiến hành kiểm thử.

	model	train_acc	test_acc	train_roc_auc	test_roc_auc
0	Bernoulli	0.883228	0.813160	0.883192	0.813990
1	Logistic Regression - [solver: liblinear]	0.895987	0.801196	0.895931	0.802337
2	Logistic Regression - [solver: newton-cg]	0.895987	0.801196	0.895931	0.802337
3	Logistic Regression - [solver: lbfgs]	0.895987	0.801196	0.895931	0.802337
4	SVC - [kernel: sigmoid]	0.879092	0.796270	0.879038	0.797405
5	SVC - [kernel: rbf]	0.959169	0.795215	0.959138	0.796428
6	SVC - [kernel: linear]	0.909275	0.792048	0.909224	0.793220

- **Nhận xét:**

- Có thể thấy các model SVC thực chất bị overfitting quá lớn.
 - Bernoulli và Logistic hoạt động ổn hơn tuy nhiên chênh lệch không lớn giữa cả hai.
- Nay ta sẽ lưu các model này cùng vectorizer TF-IDF **min_df=5** vào cùng một **SentimentModel**.

```

1 logistic_model = Model.SentimentModel(lst_tunning_models[0][1], vectorizers[3], y_train)
2 svc_model = Model.SentimentModel(lst_tunning_models[5][1], vectorizers[3], y_train)
3 bernoulli_model = Model.SentimentModel(lst_tunning_models[6][1], vectorizers[3], y_train)
4
5 print(logistic_model.info())
6 print(svc_model.info())
7 print(bernoulli_model.info())

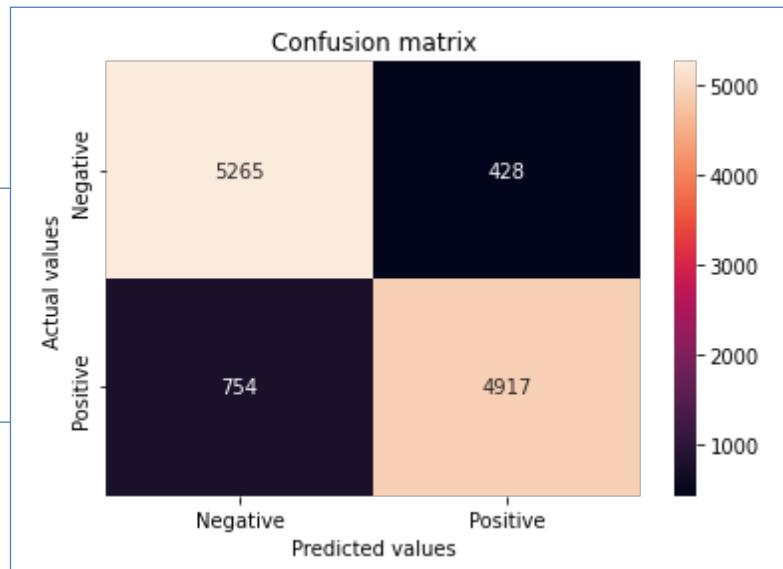
LogisticRegression(solver='liblinear')
None
SVC(kernel='sigmoid', probability=True, random_state=42)
None
BernoulliNB(alpha=0.5)
None

```

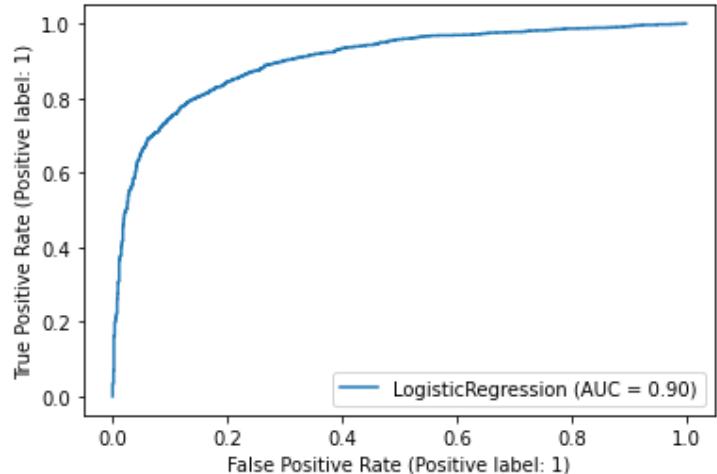
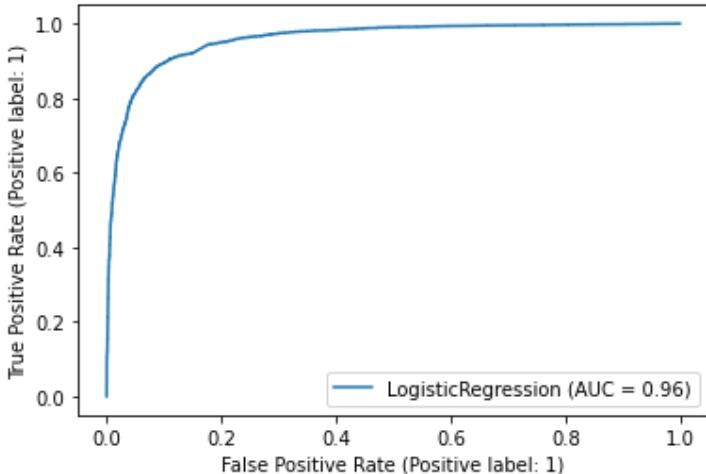
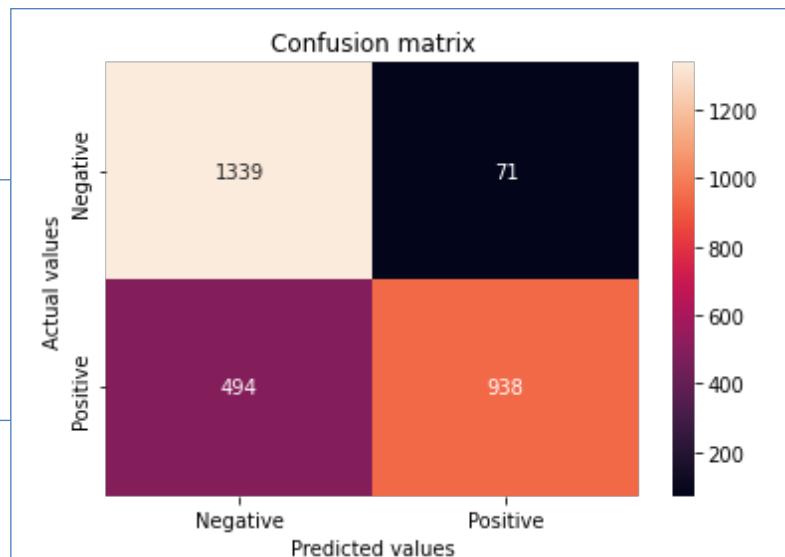
- Tiếp theo, ta sẽ đánh giá các model trên test data và training data và thực hiện trực quan hóa kết quả đánh giá.

• **Logistic model**

	precision	recall	f1-score	support
Negative	0.87	0.92	0.90	5693
Positive	0.92	0.87	0.89	5671
accuracy			0.90	11364
macro avg	0.90	0.90	0.90	11364
weighted avg	0.90	0.90	0.90	11364



	precision	recall	f1-score	support
Negative	0.73	0.95	0.83	1410
Positive	0.93	0.66	0.77	1432
accuracy			0.80	2842
macro avg	0.83	0.80	0.80	2842
weighted avg	0.83	0.80	0.80	2842



	input	output_proba	output_class
0	form_không đẹp_lấm	(0.4386474015732905, 0.5613525984267095)	1
1	áo_rộng thật_sự nhanh_chặt_lượng vải_tốt	(0.4646216101652917, 0.5353783898347083)	1
2	màu túi_hơi tối_do ánh_chụp	(0.6681474279263144, 0.33185257207368557)	0
3	chất_liệu vải_không ổn_lấm	(0.7776463727416287, 0.2223536272583713)	0
4	mã màu_đen hiện lên_áo màu_đen còn_mã đen hiện...	(0.9179684431603199, 0.08203155683968008)	0
...
11359	bán_combo ghi_đôi nhưng_nhận chỉ_đôi_tất hơi_mỏng	(0.7796936845478296, 0.22030631545217041)	0
11360	áo_đẹp from_chuẩn sẽ_ứng_hộ tiếp	(0.004412615571534051, 0.995587384428466)	1
11361	đặt_màu trắng_kem giao_màu xanh_đen giao_không...	(0.9557012480303715, 0.044298751969628535)	0
11362	đồ_chơi tí_hon_nên_dùng bán sản_phẩm như_kéo n...	(0.7663980021588869, 0.23360199784111313)	0
11363	son đẹp_lấm mọi_đáng đồng_tiền bát mịn_murót nh...	(0.22154859543527394, 0.7784514045647261)	1

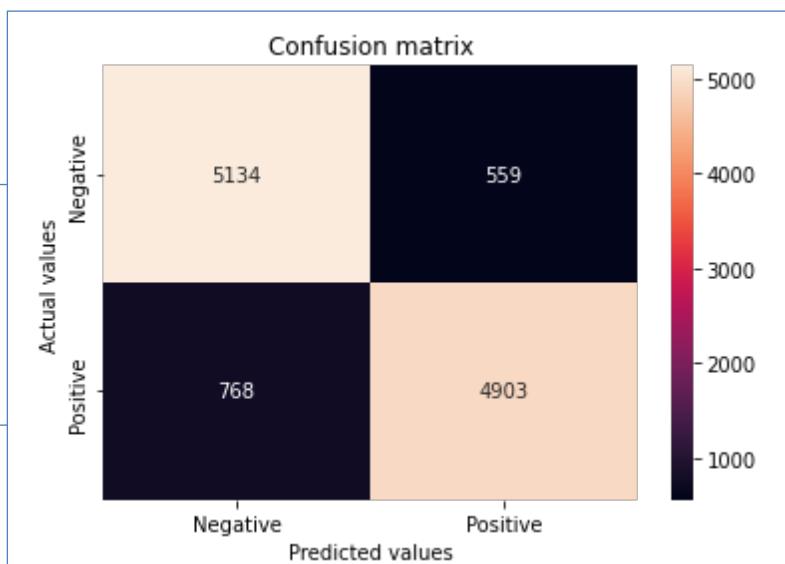
11364 rows × 3 columns

	input	output_proba	output_class
0	bán hàng kỉ quấn đầy đủ chất lượng ok đồ chuẩn...	(0.48159796534502486, 0.5184020346549751)	1
1	đặt xl áo ghi xl nhưng bé tíẹo ngang size mì...	(0.8521448671486482, 0.14785513285135182)	0
2	kẹp rất rất đẹp săn được giá sale hôm chỉ giá ...	(0.5724634666664581, 0.4275365333354197)	0
3	không nghĩ áo đẹp giao hàng nhanh không bình t...	(0.6209471455821225, 0.37905285441787756)	0
4	hôm nay mới nhận được nhưng nhìn qua thấy cũng...	(0.7044428037958854, 0.2955571962041146)	0
...
2837	đặt dây buộc tóc nhưng chỉ nhận được cái nhẫn ...	(0.8797856820895619, 0.12021431791043817)	0
2838	chất vải đẹp có vẻ mát giao hàng nhanh chóng đ...	(0.6245509356129456, 0.3754490643870544)	0
2839	dây nhin tạm không đẹpmắc sai dây phải ngồi c...	(0.8970152284934726, 0.10298477150652739)	0
2840	quần đẹpcátlượng oknhưng chữ của quần màu ...	(0.5266206280660491, 0.47337937193395085)	0
2841	ok hơi chật tí	(0.3927679458414448, 0.6072320541585552)	1

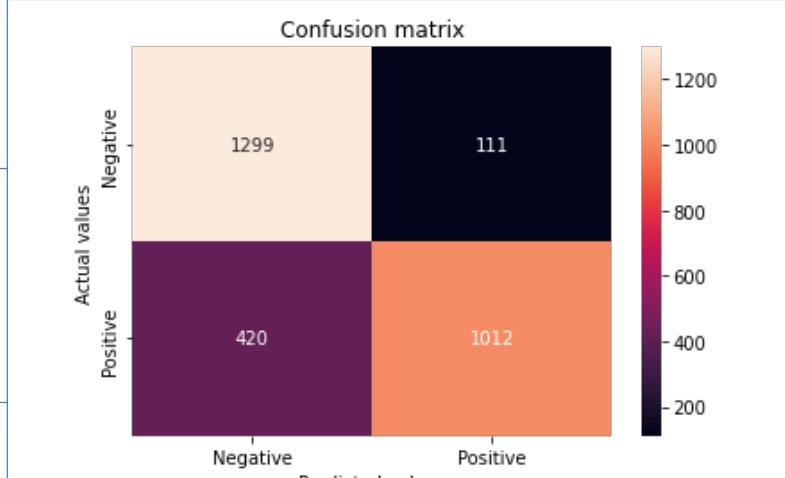
2842 rows × 3 columns

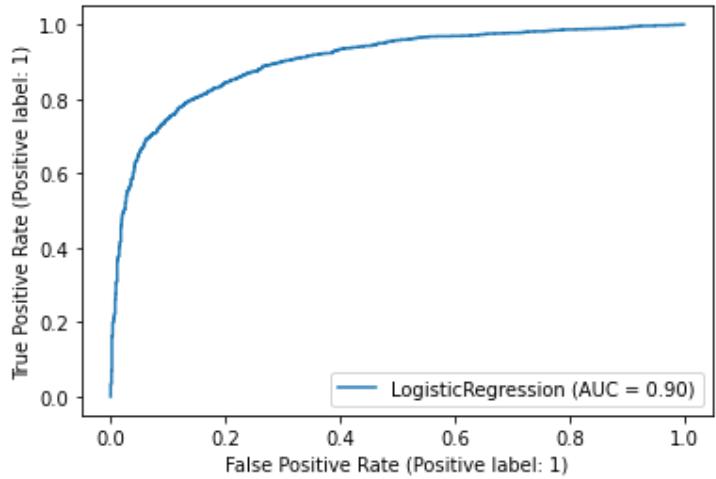
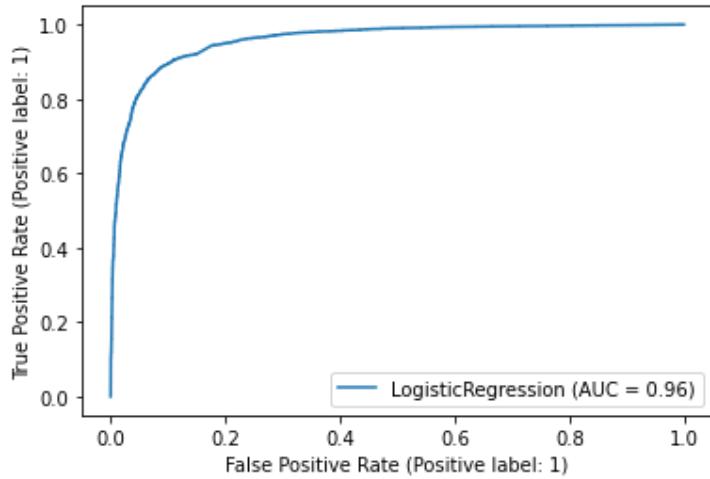
- Bernoulli model

	precision	recall	f1-score	support
Negative	0.87	0.90	0.89	5693
Positive	0.90	0.86	0.88	5671
accuracy			0.88	11364
macro avg	0.88	0.88	0.88	11364
weighted avg	0.88	0.88	0.88	11364



	precision	recall	f1-score	support
Negative	0.87	0.92	0.90	5693
Positive	0.92	0.87	0.89	5671
accuracy			0.90	11364
macro avg	0.90	0.90	0.90	11364
weighted avg	0.90	0.90	0.90	11364





	input	output_proba	output_class
0	form_không đẹp_lấm	(0.27938852559819094, 0.7206114744018083)	1
1	áo_rộng thật_sự nhanh_chặt_lượng vải_tốt	(0.5550811430542377, 0.4449188569457607)	0
2	màu túi_hơi tối_do ánh_chụp	(0.6092158558679219, 0.39078414413207957)	0
3	chất_liệu vải_không ổn_lấm	(0.8389786109199422, 0.16102138908005956)	0
4	mã màu_đen hiện lên_áo màu_đen còn_mã đen hiện...	(0.9999259433148628, 7.405668513621642e-05)	0
...
11359	bán_combo ghi_đôi nhưng nhận chỉ_đôi_tất hơi_mỏng	(0.9542953245194388, 0.04570467548056075)	0
11360	áo_đẹp from_chuẩn sē ứng_hộ tiếp	(1.4126967233954881e-05, 0.9999858730327645)	1
11361	đặt_màu trắng_kem giao_màu xanh_đen giao_không...	(0.9999999961368786, 3.863115976991816e-09)	0
11362	đồ_chơi tí_hon_nên_dùng bán sản_phẩm như_kéo n...	(0.9995662767959296, 0.00043372320407211317)	0
11363	son đẹp_lấm mọi_đáng đồng tiền bát mịn_mướt nh...	(0.0012475979230863886, 0.9987524020769104)	1

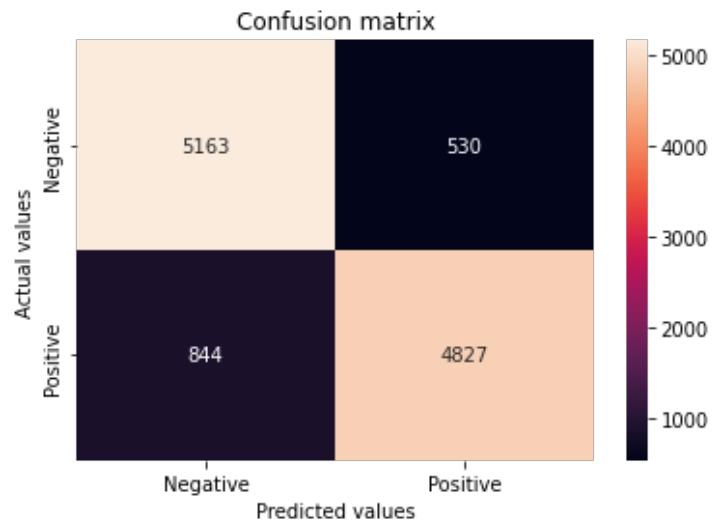
11364 rows × 3 columns

	input	output_proba	output_class
0	bán hàng kỉ quấn đầy đủ chất lượng ok đồ chuẩn...	(0.2775253204745166, 0.7224746795254852)	1
1	đặt xl áo ghi xl nhưng bé tí tẹo ngang size mì...	(0.999737308147709, 0.0002626918522946248)	0
2	kèp rất_rất đẹp săn được giá sale hôm chì giá ...	(0.06803047753757607, 0.9319695224624265)	1
3	không nghĩ áo_đẹp giao hàng nhanh không bình t...	(0.7278938523193605, 0.27210614768063934)	0
4	hôm nay mới nhận được nhưng nhìn qua thấy cũn...	(0.9991247031097783, 0.0008752968902293736)	0
...
2837	đặt dây buộc tóc nhưng chỉ nhận được cái nhẫn ...	(0.999814019575968, 0.00018598042402974397)	0
2838	chất vải_đẹp có_về mát giao hàng nhanh chóng đ...	(0.9492395884572257, 0.050760411542773554)	0
2839	dây nhin tạm không_đẹp mắc sai dây phải ngồi c...	(0.994447850343121, 0.005552149656880654)	0
2840	quần_đẹp chất_lượng ok nhưng chữ_của quần màu ...	(0.9614041467590597, 0.038595853240945284)	0
2841	ok hơi_chặt tí	(0.6975913174035353, 0.30240868259646325)	0

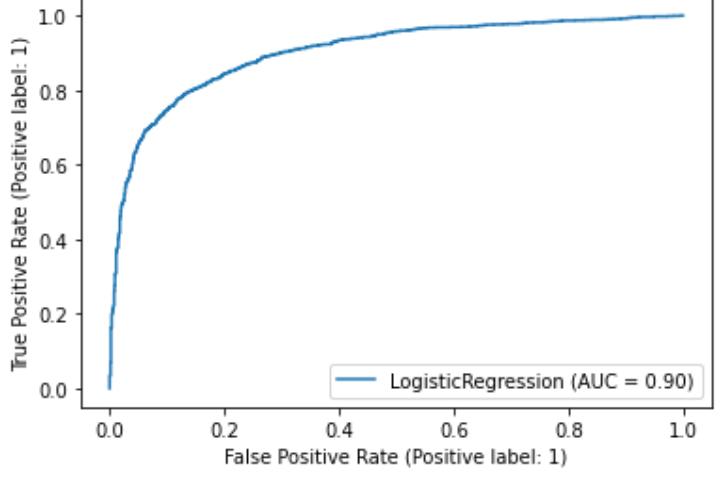
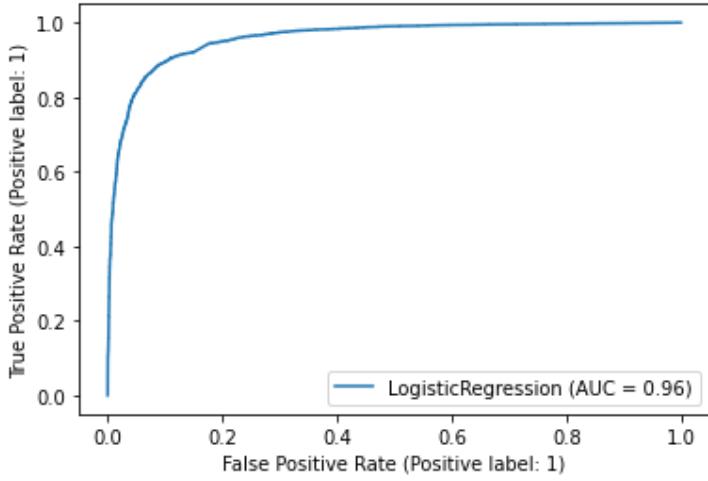
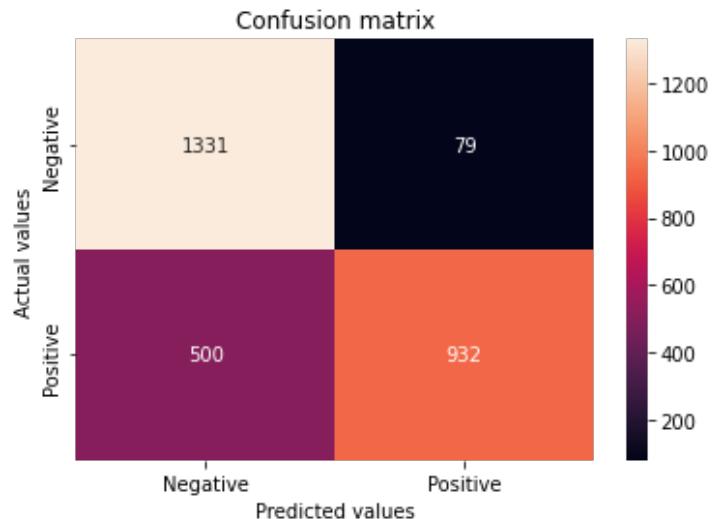
2842 rows × 3 columns

- SVC model:

	precision	recall	f1-score	support
Negative	0.86	0.91	0.88	5693
Positive	0.90	0.85	0.88	5671
accuracy			0.88	11364
macro avg	0.88	0.88	0.88	11364
weighted avg	0.88	0.88	0.88	11364



	precision	recall	f1-score	support
Negative	0.73	0.94	0.82	1410
Positive	0.92	0.65	0.76	1432
accuracy			0.80	2842
macro avg	0.82	0.80	0.79	2842
weighted avg	0.83	0.80	0.79	2842



	input	output_proba	output_class
0	form_không đẹp_lấm	(0.4416788692111604, 0.5583211307888396)	1
1	áo_rộng thật_sự nhanh_chặt_lượng vải_tốt	(0.49147172677271195, 0.5085282732272882)	0
2	màu túi_hơi tối_do ánh_chụp	(0.7810384724275545, 0.21896152757244533)	0
3	chất_liệu vải_không ổn_lấm	(0.8673607994057431, 0.13263920059425702)	0
4	mã màu_đen hiện lên_áo màu_đen còn_mã đen hiện...	(0.9559105441189036, 0.04408945588109635)	0
...
11359	bán_combo ghi_đôi nhưng nhận chỉ_đôi_tất hơi_mỏng	(0.8726656001389425, 0.12733439986105763)	0
11360	áo_đẹp from_chuẩn sẽ_ứng_hộ tiếp	(6.2369310052548105e-06, 0.9999937630689948)	1
11361	đặt_màu trắng_kem giao_màu xanh_đen giao_không...	(0.9900398854317436, 0.009960114568256377)	0
11362	đồ_chơi tí_hon_nên_dùng bán_sản_phẩm như_kéo n...	(0.846558897303596, 0.1534411026964039)	0
11363	son đẹp_lấm mọi_đáng đồng_tiền bát mịn_mướt nh...	(0.13411534842196485, 0.8658846515780353)	1

11364 rows × 3 columns

	input	output_proba	output_class
0	bán hàng kỉ quấn đầy đủ chất lượng ok đồ chuẩn...	(0.6167365115170309, 0.3832634884829691)	0
1	đặt xl áo ghi xl nhưng bé tí tẹo ngang size mì...	(0.894104170676397, 0.10589582932360307)	0
2	kèp rất_rất đẹp săn được giá sale hôm chỉ giá ...	(0.5357170371194775, 0.4642829628805225)	0
3	không nghĩ áo_đẹp giao hàng nhanh không bình t...	(0.7478781389986306, 0.25212186100136963)	0
4	hôm nay mới nhận được nhưng nhìn qua thấy cũng...	(0.8129891112243327, 0.1870108887756672)	0
...
2837	đặt dây buộc tóc nhưng chỉ nhận được cái nhẫn ...	(0.9438388548132551, 0.05616114518674483)	0
2838	chất vải đẹp có_về mát giao hàng nhanh chóng đ...	(0.7424478908159614, 0.2575521091840386)	0
2839	dây nhin tạm không đẹp mắc sai dây phải ngồi c...	(0.9674587281326442, 0.032541271867355594)	0
2840	quần đẹp chất lượng ok nhưng chữ của quần màu ...	(0.5644451750697096, 0.4355548249302903)	0
2841	ok hơi chật tí	(0.3242860704594871, 0.6757139295405128)	1

2842 rows × 3 columns

- Nhận xét:

- Có thể thấy, khả năng dự đoán còn kém trên các class positive. Nhiều comment thực chất là positive nhưng lại bị dự đoán nhầm sang negative.
- Khả năng dự đoán trên class negative tốt hơn ở các model. Điều này có thể là do variance error. Ở project một, ta đã tách các comment có 4 đến 5 sao vào nhóm positive và còn lại thuộc class negative. Tuy nhiên điều này có thể sai:
 - Thực chất có các comment 3 sao đôi khi là các khách hàng trung bình, họ không thích cũng như không ghét sản phẩm, lúc này bài toán của chúng ta có thể tách thành 3 class là **positive**, **neutral** và **negative** để hi vọng khả năng dự đoán của model tốt hơn.

- Do noise sample, lí do là có những comment mắc dù về ngữ nghĩa là positive nhưng nó lại thuộc nhóm negative và ngược lại. Tức ta không tiền xử lí dữ liệu tốt. Tuy nhiên ta khó lòng giải quyết vì nó phụ thuộc vào đánh giá chủ quan của con người. Nhưng với khả năng hiện tại nhóm khó khắc phục được lỗi này.
- Nay giờ ta sẽ lưu lại các model này cùng với vectorizer TF-IDF **min_df=5**. Ta cũng cần lưu lại file tập N-Grams các từ ghép tổ hợp từ 2, 3 và 4 từ để có thể tiến hành transform khi có dữ liệu mới.

```

1 Model.saveByPickle(svc_model, "./models/svc_model_comments.pickle")
2 Model.saveByPickle(logistic_model, "./models/logistic_model_comments.pickle")
3 Model.saveByPickle(bernoulli_model, "./models/bernoulli_model_comments.pickle")

<modules.model.SentimentModel object at 0x7f55797bc400> has been saved at ./models/svc_model_
comments.pickle.
<modules.model.SentimentModel object at 0x7f55797bc100> has been saved at ./models/logistic_m
odel_comments.pickle.
<modules.model.SentimentModel object at 0x7f55797bc280> has been saved at ./models/bernoulli_
model_comments.pickle.

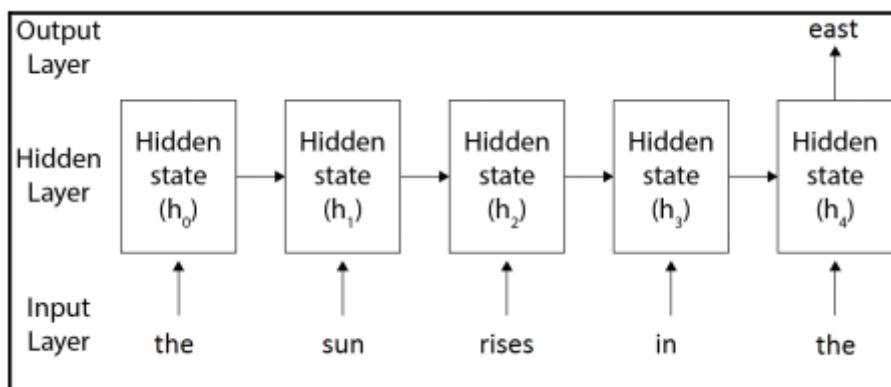
1 ngram_words.to_csv("./modules/dependencies/ngrams.csv")

```

3.2.3 Công việc 2.2 (Tiếp theo) (File 08.model.ipynb)

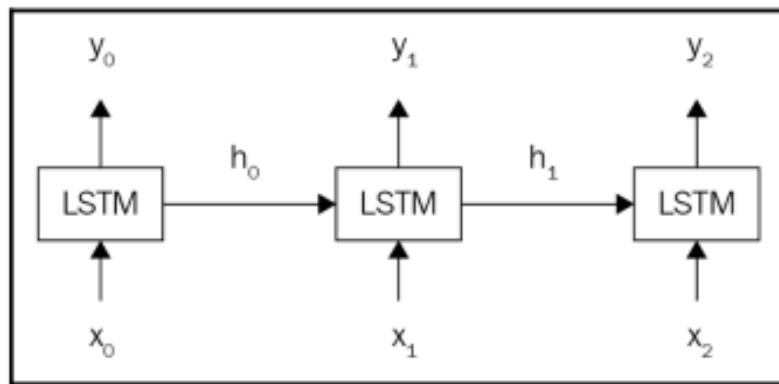
- **Nội dung:**

- Ở phần này, chúng ta sẽ xây dựng một **Comment sentiment model** bằng LSTM.
- Chúng ta biết rằng hạn chế của RNN là nó không thể lưu trữ các thông tin trong dài hạn - RNN tiến hành lưu trữ các thông tin mà nó học được từ chuỗi vào các **hidden state** nhưng khi chuỗi đầu vào quá dài - nó sẽ không thể giữ lại các thông tin này do vấn đề **gradient-vanishing** [tiêu biến gradient] trong quá trình backpropagation:

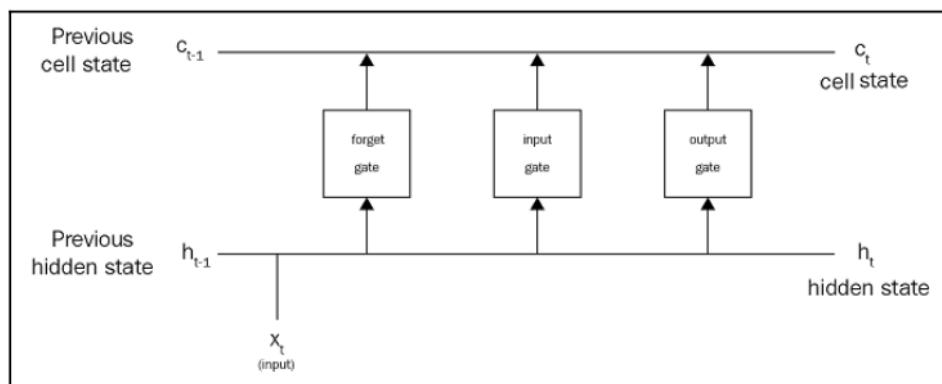


- LSTM ra đời để khắc phục gradient-vanishing bằng một cấu trúc đặc biệt gọi là **gate**. Gate sẽ lưu các thông tin của chuỗi lại đồng thời nó quản lý được thông tin nào nên xóa khỏi bộ nhớ và thông tin nào cần giữ.
- Giả sử chúng ta có câu: **Giao hàng nhanh**. Với RNN nó dễ dàng dự đoán được comment này thuộc class positive. Nhưng giả sử ta có câu dài hơn: **Giao hàng chậm "nhưng" hàng đẹp - cảm ơn shop**. Thì câu hỏi đặt ra là làm sao RNN có thể lưu trữ các thông tin trước từ

nhưng là negative word và phía sau là positive word. RNN sẽ không thể lưu trữ các thông tin này do vấn đề vanishing-gradient. Nhưng với LSTM nó sẽ giảm bớt hiện tượng này nhờ sử dụng một cấu trúc là LSTM cell.



- Lúc này, mỗi LSTM cell sẽ chứa ba gate lần lượt là:
 - **Input gate**: quyết định thông tin nào cần được thêm vào từ chuỗi đầu vào.
 - **Output gate**: quyết định thông tin nào cần được lưu trữ.
 - **Forget gate**: thông tin nào cần được loại bỏ.



- Ở RNN, một hidden state được sử dụng cho hai mục đích là lưu trữ thông tin và đưa ra dự đoán. Tuy nhiên ở LSTM nó tách hidden state thành hai phần là:
 - **Cell state**: đây là nơi lưu giữ thông tin.
 - **Hidden state**: dùng cho việc dự đoán.
- Cell state và hidden state chia sẻ thông tin lẫn nhau. Như hình trên ta thấy đầu vào sau khi đi qua hidden state → qua các gate → đến cell state.
- Bây giờ chúng ta sẽ tiến hành xây dựng một **Comment sentiment model** sử dụng LSTM của TensorFlow.



- Load dữ liệu train và test.

	raw_comment	normalize_comment	emoji_decode		label
0	form k đẹp lắm	form không đẹp lắm		0	0
1	Áo Rộng thật sự\nGi nhanh\nChất lượ...	áo rộng thật sự nhanh chất lượng v...		1	1
2	Màu túi hơi tối do với ảnh chụp	màu túi hơi tối do ảnh chụp		2	0
3	Chất liệu vải k ổn lắm	chất liệu vải không ổn lắm		3	0
4	Mã "màu đen" hiện lên áo màu đen, còn ...	mã màu đen hiện lên áo màu đen còn mã...		4	0
...
11359	Khi bán combo ghi 3 đôi, nhưng nhận chỉ ...	bán combo ghi đôi nhưng nhận chỉ đôi ta...		11359	0
11360	Áo đẹp, from chuẩn. Sẽ ủng hộ tiếp ạ !	áo đẹp from chuẩn sẽ ủng hộ tiếp		11360	1
11361	Đặt màu trắng kem thì giao màu xanh đen...	đặt màu trắng kem giao màu xanh đen giao...	expressionless	11361	0
11362	Đồ chơi tí hon.\nShop nên dừng bán sp ...	đồ chơi tí hon nên dừng bán sản phẩ...		11362	0
11363	Son đẹp lắm nha mọi ng, đáng đồng tiề...	son đẹp lắm mọi ng đáng đồng tiền bát m...		11363	1
11364 rows × 3 columns					

	raw_comment	normalize_comment	emoji_decode		label
0	Shop bán hàng kỉ quán đầy đủ. Chất l...	bán hàng kỉ quán đầy đủ chất lượ...		0	1
1	Đặt xl, áo ghi xl nhưng lại bé tí tẹo ...	đặt xl áo ghi xl nhưng bé tí tẹo ngang ...		1	0
2	Kèp rất rất đẹp săn được giá sale h...	kèp rất rất đẹp săn được giá sale h...		2	1
3	K nghĩ áo đẹp vậy đâu giao hàng nhanh k...	Không nghĩ áo đẹp giao hàng nhanh không ...		3	1
4	Hôm nay mới nhận dc nhưng nhìn qua thâ...	hôm nay mới nhận được nhưng nhìn qua...		4	1
...
2837	Minh đặt 2 dây buộc tóc nhưng lại ch...	đặt dây buộc tóc nhưng chỉ nhận đươ...	star slightly_smiling_face slightly_smiling_fa...	2837	0
2838	Chất vải đẹp, có vải mát, giao hàng nha...	chất vải đẹp có vải mát giao hàng nhanh...		2838	1
2839	Dây nhin tạm. K đẹp. Mắc sai dây mình ...	dây nhin tạm không đẹp mắc sai dây pha...		2839	0
2840	Quần đẹp chất lượng ok nhưng chữ ru...	quần đẹp chất lượng ok nhưng chữ cu...		2840	1
2841	Ok hơi chật tí	ok hơi chật tí		2841	1
2842 rows × 3 columns					

- Ở đây, nhóm đã viết gọn tất cả các tính năng của một LSTM model vào một object là **SentimentLSTM** nằm trong file **deep_model.py**, về ý tưởng của nhóm như sau:

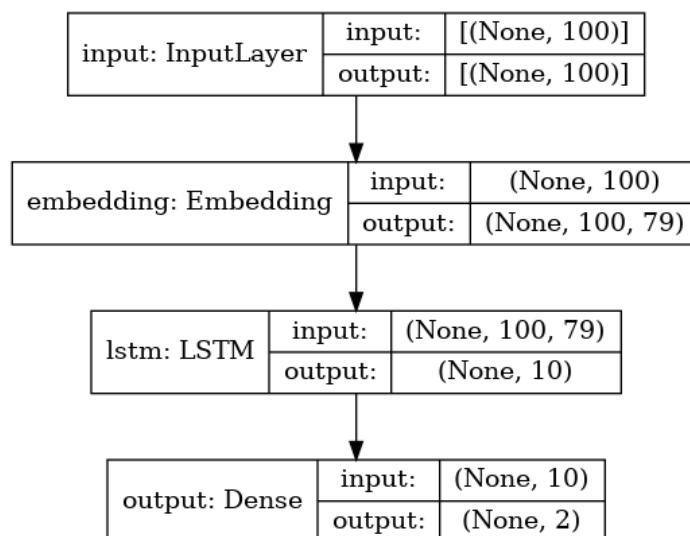
- **Bước 1:** Dữ liệu training ở đây là các comment đã qua bước tiền xử lí ở feature **normalize_comment** - ta bỏ qua khôn cần xử lí N-Grams cho feature này.
- **Bước 2:** Ta sử dụng cơ chế Tokenizer do TensorFlow cung cấp, object **Tokenizer** của TensorFlow yêu cầu một hyper-parameter là **num_words** - tức số lượng từ tối đa ta muốn tokenize, ở đây nếu để bằng **None**, **Tokenizer** sẽ được tự động tính. Cuối cùng **Tokenizer** sẽ lưu các comment dưới dạng mảng chứa các index của từ, với index của từ phổ biến nhất là 1.

- Bước 3:** Ta sử dụng phương thức **pad_sequences()** của TensorFlow - phương thức này sẽ dựa vào thuộc tính **maxlen** để đưa toàn bộ các comment về cùng một độ dài, các comment nào có chiều dài lớn sẽ được cắt bớt. Các comment ngắn hơn sẽ được thay thế bằng giá trị 0.
 - Bước 4:** Ta sẽ định nghĩa kiến trúc của model trong **SentimentLSTM._defineModel()**. Ở đây, để tiện cho quá trình HyperParams Optimization, ta sẽ để thuộc tính **optimizer** linh động dựa trên tham vào truyền vào của hàm. Tiếp theo, ở output ta sử dụng 2 neuron để lưu giá trị của xác suất để dự đoán comment đó ở lần lượt hai class là negative và positive, ta sử dụng loss function là **binary cross-entropy** và action function **softmax** cho output layer này.
 - Bước 5:** Để ta có thể kiểm soát được liệu model có bị overfitting trong quá trình training không, nếu có ta có thể dừng quá trình training lại. Tại đây ta sử dụng object **CheckPoint** do TensorFlow cung cấp, object này sẽ theo dõi qua trình training và dựa vào thông số trên loss function mà ta chỉ định là trên train hay validation data để tiến hành lưu lại **weights set** của model tại thời điểm loss value này thấp nhất, ở đây nhóm sẽ sử dụng 10% dữ liệu training làm validation và chỉ định validation loss để tránh overfitting.
- Code dưới đây ta sẽ định nghĩa một hyperparams set cho model. Ở đây **num_words** và **maxlen** là **None** để cho TensorFlow tự tính toán, ta sử dụng 10 neuron cho LSTM layer và dropout regularization có xác suất để một neuron bị tắt ở layer này là 20%.
- Sau đó ta sẽ cho in **summary** để xem tổng số weight của model và trực quan hóa kiến trúc của model.

```
Model: "sequential"
```

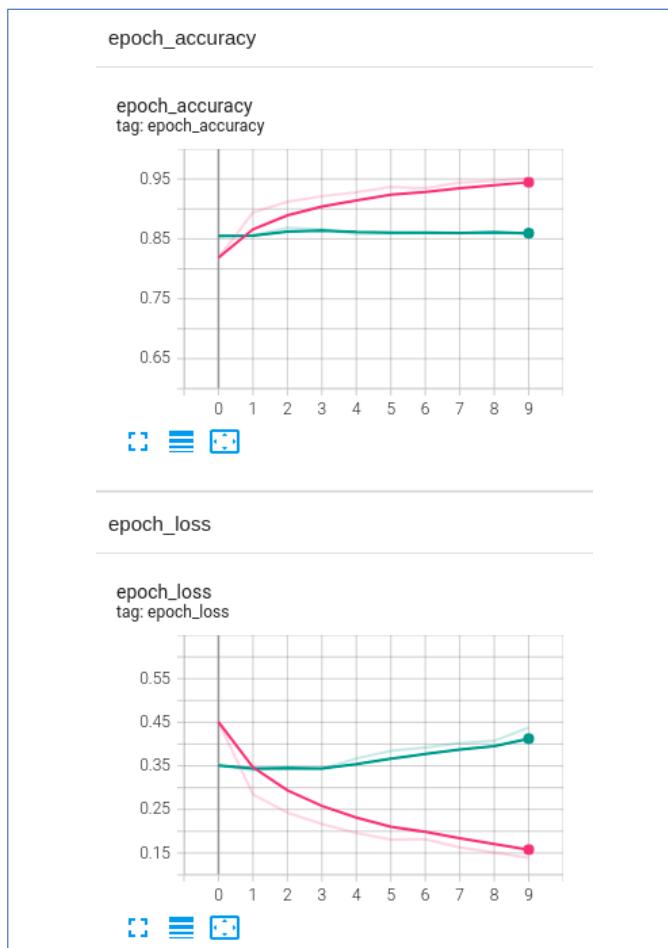
Layer (type)	Output Shape	Param #
<hr/>		
embedding (Embedding)	(None, 100, 100)	358500
lstm (LSTM)	(None, 10)	4440
output (Dense)	(None, 2)	22
<hr/>		
Total params: 362,962		
Trainable params: 362,962		
Non-trainable params: 0		

```
('You must install pydot (`pip install pydot`) and install graphviz (see
instructions at https://graphviz.gitlab.io/download/) ', 'for plot_
mode
l/model_to_dot to work.')
```



- Bây giờ ta tiến hành training model.

```
320/320 [=====] - 31s 86ms/step - loss: 0.4508 - accuracy: 0.8185 -  
val_loss: 0.3506 - val_accuracy: 0.8549  
Epoch 2/10  
320/320 [=====] - 28s 86ms/step - loss: 0.2847 - accuracy: 0.8943 -  
val_loss: 0.3400 - val_accuracy: 0.8558  
Epoch 3/10  
320/320 [=====] - 26s 80ms/step - loss: 0.2424 - accuracy: 0.9119 -  
val_loss: 0.3460 - val_accuracy: 0.8681  
Epoch 4/10  
320/320 [=====] - 25s 79ms/step - loss: 0.2164 - accuracy: 0.9212 -  
val_loss: 0.3432 - val_accuracy: 0.8663  
Epoch 5/10  
320/320 [=====] - 26s 80ms/step - loss: 0.1960 - accuracy: 0.9278 -  
val_loss: 0.3669 - val_accuracy: 0.8584  
Epoch 6/10  
320/320 [=====] - 26s 82ms/step - loss: 0.1812 - accuracy: 0.9369 -  
val_loss: 0.3840 - val_accuracy: 0.8593  
Epoch 7/10  
320/320 [=====] - 14s 44ms/step - loss: 0.1814 - accuracy: 0.9343 -  
val_loss: 0.3922 - val_accuracy: 0.8602  
Epoch 8/10  
320/320 [=====] - 14s 44ms/step - loss: 0.1626 - accuracy: 0.9439 -  
val_loss: 0.4019 - val_accuracy: 0.8593  
Epoch 9/10  
320/320 [=====] - 14s 44ms/step - loss: 0.1515 - accuracy: 0.9471 -  
val_loss: 0.4074 - val_accuracy: 0.8619  
Epoch 10/10  
320/320 [=====] - 14s 44ms/step - loss: 0.1386 - accuracy: 0.9514 -  
val_loss: 0.4378 - val_accuracy: 0.8575
```



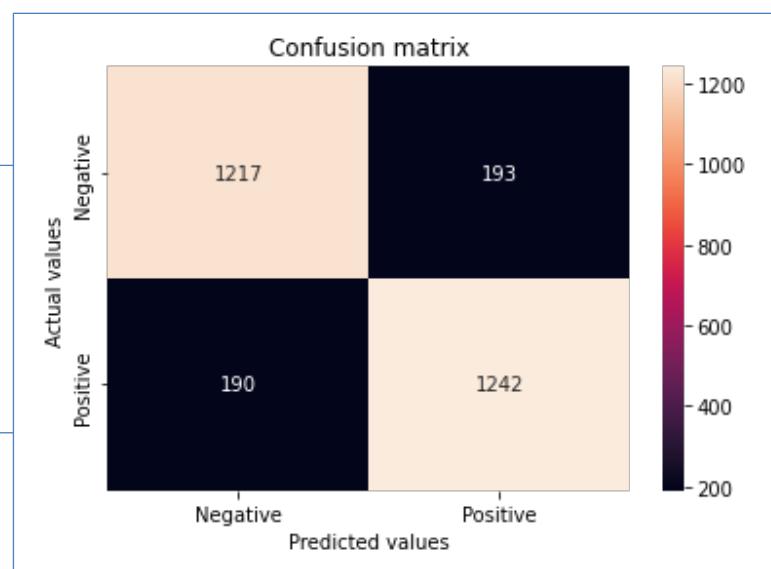
- **Nhận xét:**

- Hai biểu đồ trên lần lượt thể hiện giá trị accuracy và loss value qua 10 epoch với đường màu xanh là của **validation data** và hồng là của **training data** được vẽ bằng TensorBoard. Có thể thấy kể từ epoch thứ 2 trở đi thì loss value có xu hướng đi xuống ở training data và đi lên ở validation. Lúc này nhờ cơ chế CheckPoint của TensorFlow mà các weight tại thời điểm này được lưu lại và ta có thể kiểm tra sau này.
- Tiếp theo, ta sẽ tokenizer cho test data để tiến hành đánh giá trên test data.

	input	output_proba	output_class
0	bán hàng kỉ quần đầy đủ chất lượng...	(4.148319e-06, 0.9999958)	1
1	đặt xl áo ghi xl nhưng bé tíẹo ngang ...	(0.9960705, 0.003929518)	0
2	kèp rất rất đẹp săn được giá sale h...	(0.809507, 0.19049297)	0
3	không nghĩ áo đẹp giao hàng nhanh không ...	(0.22817881, 0.7718212)	1
4	hôm nay mới nhận được nhưng nhìn qua...	(0.99997556, 2.4480925e-05)	0
...
2837	đặt dây buộc tóc nhưng chỉ nhận đượ...	(0.9999796, 2.0352623e-05)	0
2838	chất vải đẹp có vẻ mát giao hàng nhanh...	(0.0008592206, 0.99914074)	1
2839	dây nhin tạm không đẹp mắc sai dây pha...	(0.9999949, 5.076978e-06)	0
2840	quần đẹp chất lượng ok nhưng chữ cu...	(0.9882366, 0.011763366)	0
2841	ok hơi chật tí	(0.0049098604, 0.9950901)	1

2842 rows × 3 columns

	precision	recall	f1-score	support
Negative	0.86	0.86	0.86	1410
Positive	0.87	0.87	0.87	1432
accuracy			0.87	2842
macro avg	0.87	0.87	0.87	2842
weighted avg	0.87	0.87	0.87	2842



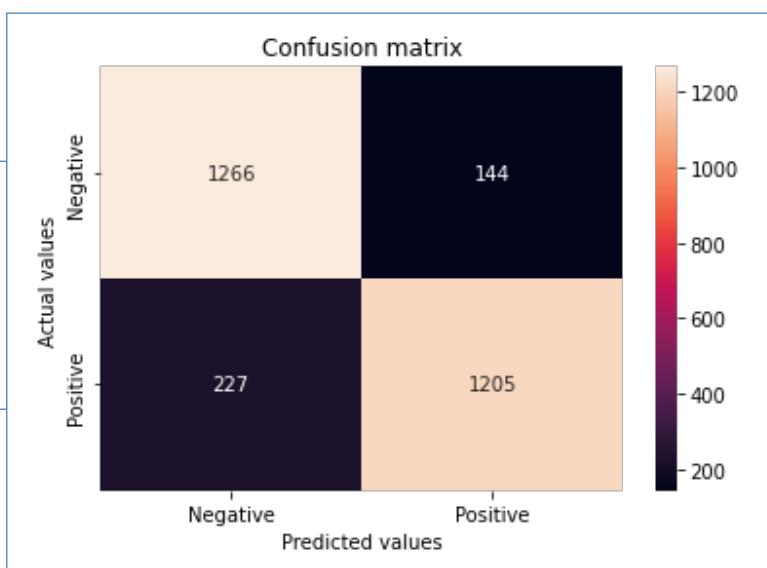
- **Nhận xét:**

- Ta có thể thấy rõ ràng rằng dù độ chính xác của model LSTM khá cao hơn 85%, với việc để các hyper-params cho TensorFlow tự quyết thì model có khả năng dự đoán cân bằng trên cả hai class là negative và positive, ta không cần phải kiểm tra lại bằng ROC-AUC.
- Do ta có áp dụng checkpoint trong model để lưu lại các bộ tham số tốt nhất trong quá trình training, ta có thể lấy bộ tham số mà có validation loss thấp nhất để update trọng số cho model.

	input	output_proba	output_class
0	bán hàng kỉ quần đầy đủ chất lượng...	(0.00073411025, 0.99926585)	1
1	đặt xl áo ghi xl nhưng bé tí tẹo ngang ...	(0.9971385, 0.002861474)	0
2	kèp rất rất đẹp săn được giá sale h...	(0.9313881, 0.068611994)	0
3	không nghĩ áo đẹp giao hàng nhanh không ...	(0.011878509, 0.9881215)	1
4	hôm nay mới nhận được nhưng nhìn qua...	(0.9889897, 0.011010231)	0
...
2837	đặt dây buộc tóc nhưng chỉ nhận đượ...	(0.9959667, 0.0040333555)	0
2838	chất vải đẹp có vẻ mát giao hàng nhanh...	(0.010884253, 0.9891158)	1
2839	dây nhìn tạm không đẹp mắc sai dây pha...	(0.99751866, 0.002481401)	0
2840	quần đẹp chất lượng ok nhưng chữ cu...	(0.91258466, 0.08741527)	0
2841	ok hơi chật tí	(0.23734112, 0.7626589)	1

2842 rows × 3 columns

	precision	recall	f1-score	support
Negative	0.85	0.90	0.87	1410
Positive	0.89	0.84	0.87	1432
accuracy			0.87	2842
macro avg	0.87	0.87	0.87	2842
weighted avg	0.87	0.87	0.87	2842

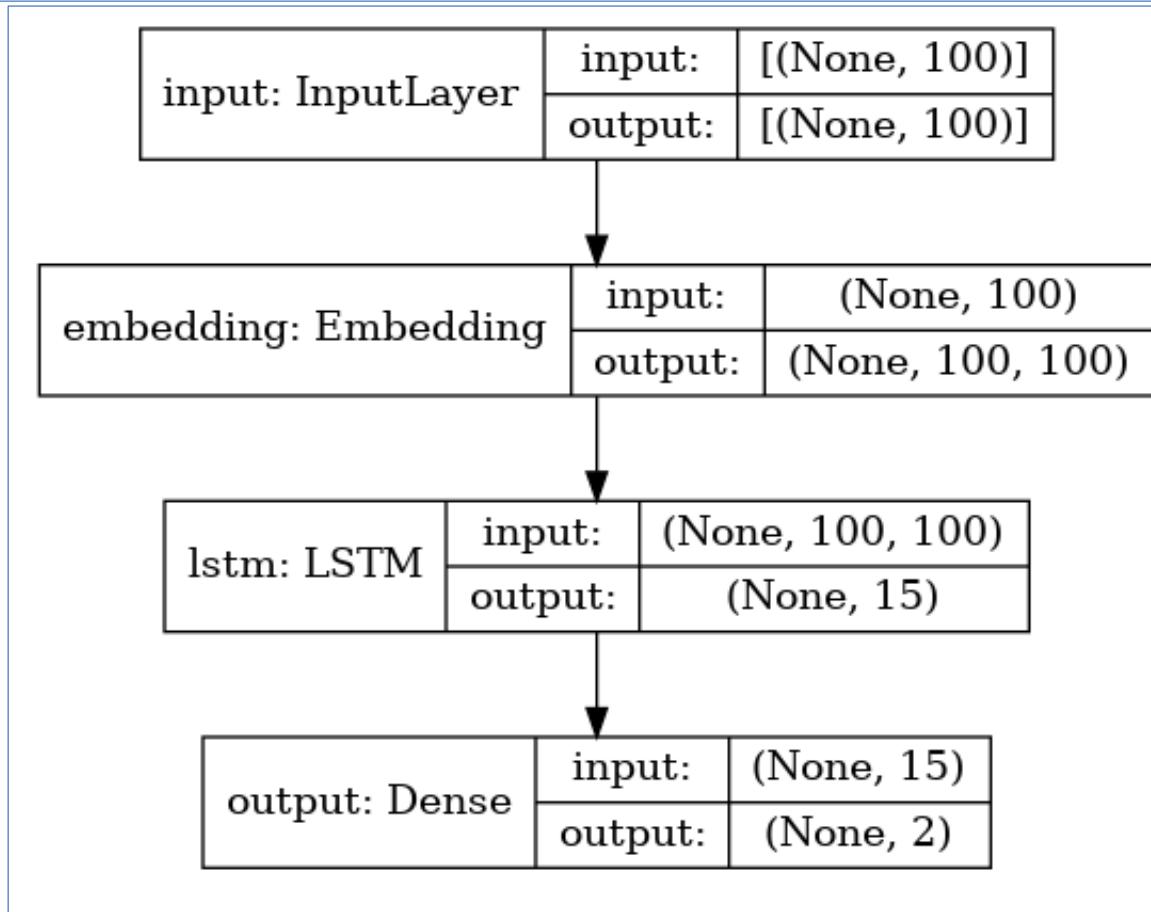


- **Nhận xét:**

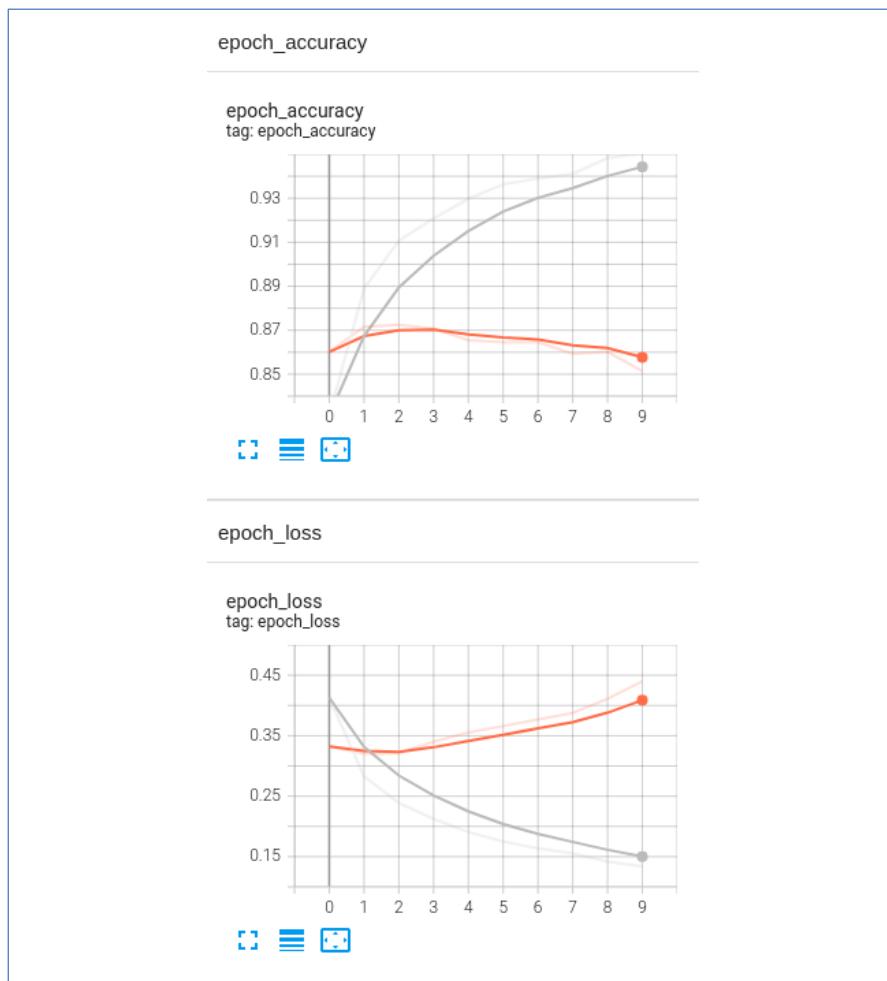
- Ta thấy rằng nếu nhìn vào các chỉ số như **precision**, **recall** và **F1-score** thì model được update bởi weight ở tại epoch thứ hai này hoạt động cho hiệu suất thậm chí tốt hơn một ít dù chỉ mới trải qua 2 epoch.
- Ta sẽ đào tạo một model thứ hai với các hyper-params là **num_word** và **maxlen** lần lượt là 3000 và 100, ở project hai ta đã tính được toàn bộ data của ta có khoảng loanh quanh 3000 từ và các comment hay có độ dài là 100 và sử dụng 15 neuron ở LSTM layer.
- Ta sẽ định nghĩa model sau đó cho training, dự đoán và đánh giá.

Model: "sequential_2"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 100)	358500
lstm (LSTM)	(None, 15)	6960
output (Dense)	(None, 2)	32
<hr/>		
Total params: 365,492		
Trainable params: 365,492		
Non-trainable params: 0		



```
Epoch 1/10
320/320 [=====] - 29s 81ms/step - loss: 0.4130 - accuracy: 0.8312 -
val_loss: 0.3323 - val_accuracy: 0.8602
Epoch 2/10
320/320 [=====] - 25s 79ms/step - loss: 0.2828 - accuracy: 0.8889 -
val_loss: 0.3205 - val_accuracy: 0.8716
Epoch 3/10
320/320 [=====] - 25s 77ms/step - loss: 0.2386 - accuracy: 0.9107 -
val_loss: 0.3210 - val_accuracy: 0.8725
Epoch 4/10
320/320 [=====] - 24s 76ms/step - loss: 0.2124 - accuracy: 0.9209 -
val_loss: 0.3401 - val_accuracy: 0.8707
Epoch 5/10
320/320 [=====] - 25s 77ms/step - loss: 0.1907 - accuracy: 0.9298 -
val_loss: 0.3552 - val_accuracy: 0.8654
Epoch 6/10
320/320 [=====] - 24s 74ms/step - loss: 0.1752 - accuracy: 0.9363 -
val_loss: 0.3658 - val_accuracy: 0.8646
Epoch 7/10
320/320 [=====] - 23s 72ms/step - loss: 0.1636 - accuracy: 0.9390 -
val_loss: 0.3769 - val_accuracy: 0.8646
Epoch 8/10
320/320 [=====] - 23s 73ms/step - loss: 0.1552 - accuracy: 0.9411 -
val_loss: 0.3877 - val_accuracy: 0.8593
Epoch 9/10
320/320 [=====] - 24s 76ms/step - loss: 0.1420 - accuracy: 0.9483 -
val_loss: 0.4108 - val_accuracy: 0.8602
Epoch 10/10
320/320 [=====] - 25s 78ms/step - loss: 0.1337 - accuracy: 0.9505 -
val_loss: 0.4399 - val_accuracy: 0.8514
```



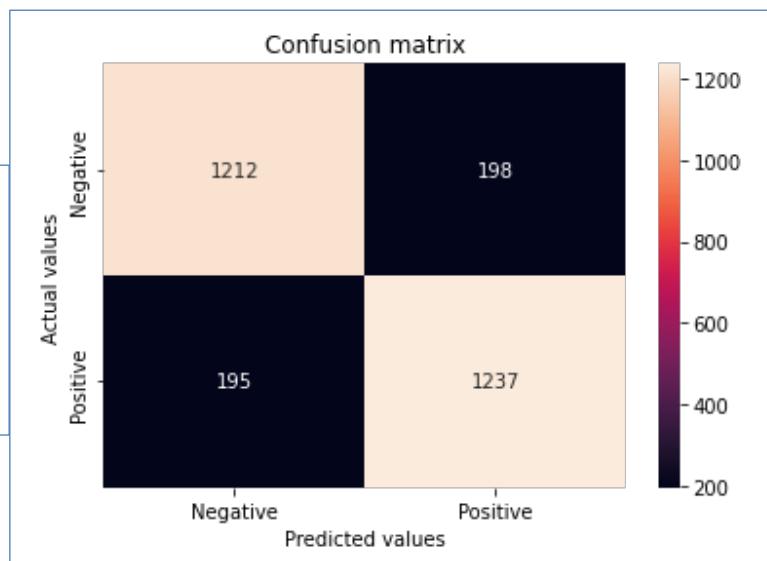
- Nhận xét:

- Với việc ta tự định nghĩa hyper-params set, kết quả đem lại không tốt hơn so với model 1. Có thể thấy rõ accuracy và loss validation trên training data (đường màu xám) và validation data (đường màu cam) bắt đầu khác nhau và khoảng cách khác biệt là lớn - không được ổn định bằng model 1.
- Ta có thể update weight tại epoch thứ 2 xem có cải thiện hơn không.

		input	output_proba	output_class
0	bán hàng kỉ quấn đầy đủ chất lượng...	(7.5561167e-07, 0.9999993)		1
1	đặt xl áo ghi xl nhưng bé tí tẹo ngang ...	(0.92751795, 0.07248204)		0
2	kèp rất rất đẹp săn được giá sale h...	(0.07028565, 0.9297143)		1
3	không nghĩ áo đẹp giao hàng nhanh không ...	(0.13486566, 0.8651343)		1
4	hôm nay mới nhận được nhưng nhìn qua...	(0.9999882, 1.1852359e-05)		0
...
2837	đặt dây buộc tóc nhưng chỉ nhận đươ...	(0.9999896, 1.0315402e-05)		0
2838	chất vải đẹp có vẻ mát giao hàng nhanh...	(0.00067545846, 0.99932456)		1
2839	dây nhìn tạm không đẹp mắc sai dây pha...	(0.99999, 1.001526e-05)		0
2840	quần đẹp chất lượng ok nhưng chữ cu...	(0.981992, 0.018007934)		0
2841	ok hơi chật tí	(0.0009869847, 0.99901295)		1

2842 rows × 3 columns

	precision	recall	f1-score	support
Negative	0.86	0.86	0.86	1410
Positive	0.86	0.86	0.86	1432
accuracy			0.86	2842
macro avg	0.86	0.86	0.86	2842
weighted avg	0.86	0.86	0.86	2842



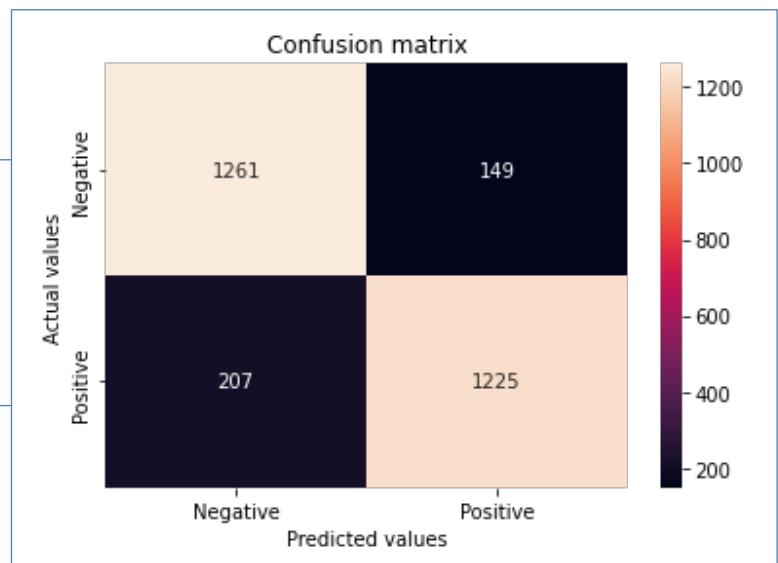
- Nhận xét:

- Độ chính xác của model 2 này là 86 phần trăm.
- Nhưng ta nên đặt câu hỏi vì sao đã qua 2 model có kiến trúc khác nhau rất nhiều nhưng độ chính xác vẫn loanh quanh 86, 87%.

	input	output_proba	output_class
0	bán hàng kỉ quần đầy đủ chất lượng...	(0.00043627014, 0.99956375)	1
1	đặt xl áo ghi xl nhưng bé tíẹo ngang ...	(0.9958728, 0.004127209)	0
2	kẹp rất rất đẹp săn được giá sale h...	(0.115075, 0.88492507)	1
3	không nghĩ áo đẹp giao hàng nhanh không ...	(0.012311526, 0.9876885)	1
4	hôm nay mới nhận được nhưng nhìn qua...	(0.8120184, 0.18798164)	0
...
2837	đặt dây buộc tóc nhưng chỉ nhận đươ...	(0.997141, 0.0028590155)	0
2838	chất vải đẹp có vẻ mát giao hàng nhanh...	(0.0035439562, 0.99645597)	1
2839	dây nhìn tạm không đẹp mắc sai dây pha...	(0.99527293, 0.004727039)	0
2840	quần đẹp chất lượng ok nhưng chữ cu...	(0.74641526, 0.2535847)	0
2841	ok hơi chật tí	(0.1930916, 0.80690837)	1

2842 rows × 3 columns

	precision	recall	f1-score	support
Negative	0.86	0.89	0.88	1410
Positive	0.89	0.86	0.87	1432
accuracy			0.87	2842
macro avg	0.88	0.87	0.87	2842
weighted avg	0.88	0.87	0.87	2842



- **Nhận xét:**

- Độ chính xác lúc này 87%.
 - Nếu ta đào tạo quá 10 epoch thì càng ngày bị overfitting, nhưng nếu ta thay đổi kiến trúc của model thì độ chính xác luôn loanh quanh 86, 87%. Như vậy đây có phải là độ chính xác tối đa mà model LSTM có thể đạt được hoặc là do chính data của ta bị variance error do có sự chồng chéo data giữa negative và positive ở các comment 3 sao.
- Bây giờ ta sẽ lưu lại các model này thành file “*.h5” và các vectorizer tương ứng thành file “*.pickle”.
- Bây giờ chúng ta sẽ tiến hành Hyper Parameters bằng phương pháp HyperBand được đề xuất vào năm 2016. Đây là bài báo đó (<https://arxiv.org/pdf/1603.06560.pdf>). Ý tưởng của phương pháp này là HyperBand sẽ random ra một vài hyper-params set ban đầu. Tiếp theo nó sẽ tiến hành đào tạo các model này và thực hiện đánh giá dựa trên loss value của train hoặc test do ta quy định. Tiếp theo nó sẽ tiến hành chọn lọc dựa trên các công thức để chọn ra các hyperparam set có loss thấp nhất và loại bỏ đi các hyperparams set cho hiệu suất kém. Với từng hyperparams set được chọn lọc này nó sẽ đào tạo lại và tiếp tục loại bỏ và đào tạo lại cho đến khi còn lại model tốt nhất.
- Bài viết gốc của tác giả đọc sẽ khó hiểu và nặng về toán học, nên ta có thể đọc tại đây để hiểu hơn (<https://medium.com/data-from-the-trenches/a-slightly-better-budget-allocation-for-hyperband-bbd45af14481>).
- Bây giờ ta sẽ tạo một hàm `get_params()` là một Random-Search ra một hyperparam-set. Tiếp theo hàm `try_params()` sẽ nhận vào hyperparam set phát sinh từ `get_params()` và tiến hành training. Hai hàm này sẽ được bô vào một object **Hyperband** như là các tham số, là object này sẽ thực hiện tính toán và trả ra hyperparams set tốt nhất.
- **Nhận xét:**

- Ở đây nhóm đã tiến hành ngắn quá trình đào tạo lại, hình dưới đây được chụp từ file `hyperband.txt` - file này ghi lại lịch sử của quá trình optimization, ta thấy rằng:

```

Epoch 1/10
WARNING:tensorflow:Model was constructed with shape (None, 100) for input KerasTensor(type_spec=TensorSpec(shape=(None, 100), dtype=tf.float32, name='input'), name='ir
WARNING:tensorflow:Model was constructed with shape (None, 100) for input KerasTensor(type_spec=TensorSpec(shape=(None, 100), dtype=tf.float32, name='input'), name='ir
228/228 [=====] - ETA: 0s - loss: 0.4778 - accuracy: 0.8154WARNING:tensorflow:Model was constructed with shape (None, 100) for input KerasTen
228/228 [=====] - 24s 100ms/step - loss: 0.4778 - accuracy: 0.8154 val_loss: 0.3681 - val_accuracy: 0.8531
Epoch 2/10
228/228 [=====] - 22s 96ms/step - loss: 0.3015 - accuracy: 0.8870 - val_loss: 0.3429 - val_accuracy: 0.8610
Epoch 3/10
228/228 [=====] - 22s 95ms/step - loss: 0.2565 - accuracy: 0.9088 - val_loss: 0.3356 - val_accuracy: 0.8637
Epoch 4/10
228/228 [=====] - 22s 97ms/step - loss: 0.2289 - accuracy: 0.9164 - val_loss: 0.3421 - val_accuracy: 0.8610
Epoch 5/10
228/228 [=====] - 22s 95ms/step - loss: 0.2162 - accuracy: 0.9193 - val_loss: 0.3563 - val_accuracy: 0.8584
Epoch 6/10
228/228 [=====] - 22s 95ms/step - loss: 0.1988 - accuracy: 0.9264 - val_loss: 0.3623 - val_accuracy: 0.8619
Epoch 7/10
228/228 [=====] - 22s 97ms/step - loss: 0.1896 - accuracy: 0.9293 - val_loss: 0.3867 - val_accuracy: 0.8628
Epoch 8/10
228/228 [=====] - 22s 95ms/step - loss: 0.1748 - accuracy: 0.9365 - val_loss: 0.4037 - val_accuracy: 0.8584
Epoch 9/10
228/228 [=====] - 22s 95ms/step - loss: 0.1726 - accuracy: 0.9356 - val_loss: 0.3851 - val_accuracy: 0.8646
Epoch 10/10
228/228 [=====] - 22s 96ms/step - loss: 0.1622 - accuracy: 0.9397 - val_loss: 0.3945 - val_accuracy: 0.8619
36/36 [=====] - 0s 11ms/step - loss: 0.1534 - accuracy: 0.9481

222 seconds.

21 | Fri Oct 29 21:43:15 2021 | lowest loss so far: 0.1051 (run 19)

Epoch 1/10
WARNING:tensorflow:Model was constructed with shape (None, 100) for input KerasTensor(type_spec=TensorSpec(shape=(None, 100), dtype=tf.float32, name='input'), name='ir
WARNING:tensorflow:Model was constructed with shape (None, 100) for input KerasTensor(type_spec=TensorSpec(shape=(None, 100), dtype=tf.float32, name='input'), name='ir
410/410 [=====] - ETA: 0s - loss: 0.3823 - accuracy: 0.8378WARNING:tensorflow:Model was constructed with shape (None, 100) for input KerasTen
410/410 [=====] - 33s 76ms/step - loss: 0.3823 - accuracy: 0.8378 - val_loss: 0.3331 - val_accuracy: 0.8628
Epoch 2/10
410/410 [=====] - 30s 74ms/step - loss: 0.2686 - accuracy: 0.8959 - val_loss: 0.3288 - val_accuracy: 0.8672
Epoch 3/10
410/410 [=====] - 30s 74ms/step - loss: 0.2306 - accuracy: 0.9139 - val_loss: 0.3267 - val_accuracy: 0.8663
Epoch 4/10
410/410 [=====] - 30s 74ms/step - loss: 0.2063 - accuracy: 0.9235 - val_loss: 0.3392 - val_accuracy: 0.8698
Epoch 5/10
410/410 [=====] - 30s 74ms/step - loss: 0.1862 - accuracy: 0.9323 - val_loss: 0.3532 - val_accuracy: 0.8619
Epoch 6/10
410/410 [=====] - 30s 74ms/step - loss: 0.1645 - accuracy: 0.9394 - val_loss: 0.3831 - val_accuracy: 0.8654
Epoch 7/10
410/410 [=====] - 30s 74ms/step - loss: 0.1573 - accuracy: 0.9405 - val_loss: 0.4077 - val_accuracy: 0.8584
Epoch 8/10
410/410 [=====] - 30s 74ms/step - loss: 0.1415 - accuracy: 0.9500 - val_loss: 0.3938 - val_accuracy: 0.8602

```

- Dù trải qua 20 lần đào tạo lại các model dựa trên các hyperparam-set khác nhau cho validation loss thấp nhất, nhưng giá trị này vẫn lớn hơn 0.3 tương tự như **model_1** và **model_2** mà ta đào tạo ban đầu, xem ra đây có thể là hiệu suất tốt nhất mà model LSTM của ta có thể làm được.
- Từ đây góp phần cho ta chắc chắn hơn rằng dataset comments của chúng ta đang thực sự bị variance error.

3.2.4 Sử dụng CNN1D của Keras (File 09.model.ipynb)

- Nội dung:

- Phần này chúng ta sẽ thử sử dụng lại **word vectors** của Facebook cung cấp mà chúng ta đã dùng để trực quan hóa dữ liệu lên không gian 3 chiều, mục đích ta muốn biết liệu các **word vectors** này có hoạt động hiệu quả trên dataset của chúng ta không.
- Ở CNN, chúng ta sử dụng phép **convolution** [tích chập] giữa **window** [cửa sổ lọc] có kích thước 3×3 , 5×5 - duyệt qua từng vùng tương ứng trên ảnh để tìm hiểu các đặc trưng phức tạp của hình ảnh.
- Vậy giả sử từng comment bao gồm n từ, ta có thể dùng **fasttext 100 dimensions cho một từ** mà ta đã làm ở project 2, như vậy lúc này một comment của chúng ta được chuyển đổi thành một ma trận có kích thước $100 \times n$.
- Nhưng lúc này các comment của chúng ta sẽ không thống nhất về chiều dài, nên ta sẽ ép chúng về chung một chiều dài m nào đó và. Và với những comment có $n < m$, ta sẽ thay thế phần bù còn lại là một vector 0.
- Khi áp dụng CNN cho bài toán NLP, người ta hay áp dụng window có kích thước từ 2 đến 5. Giả sử window có kích thước là 5 thì chúng ta đang trượt qua comment và tính toán cho 5 từ một lúc, nó hoạt động như cách chúng ta code cho N-Grams ở project 2 vậy.
- Về mặt thời gian, thì CNN hoạt động nhanh hơn so với LSTM, tuy nhiên về mặt hiệu năng, khả năng cao kết quả cho ra không chính xác, vì mặc dù ta đại diện cho một từ là một vector 100 dimensions nhưng chưa có điều gì đủ để chứng minh các vector này phù hợp với tập dữ liệu của chúng ta. Nên việc ta áp dụng ở đây đơn thuần là để kiểm tra hơn là việc hi vọng nó hiệu quả.
- Như vậy chiến lược của chúng ta sẽ như sau:
 - **Bước 1:** Ta sẽ định nghĩa một chiều dài cố định cho từng comment trong training data, như project 2 ta biết được rằng một comment có khoảng gần 100 từ, như vậy lúc này một comment của chúng ta sau quá trình **Embedding** sẽ là một ma trận có kích thước 100×100 . Và ta cập nhật **weights** cho Embedding layer bằng chính các comment mà mỗi comment là một ma trận.
 - **Bước 2:** Tiếp theo, ta sẽ định nghĩa window của chúng ta có kích thước là 5, ta tiến hành convolution giữa **Input layer** và **Embedding layer**, sau đó ta cho kết quả đi qua một **MaxPooling layer** có kích thước là 5, 5 giá trị lân cận chọn ra giá trị lớn nhất.
 - **Bước 3:** Ta sử dụng một **GlobalPoolingMax** để gộp các output ở bước trước lại, việc này tương tự như ta dùng **Flatten layer** vậy.
 - **Bước 4:** Ta định nghĩa output layer gồm 2 neurons và activation function là softmax.
 - **Bước 5:** Ta compile model bằng binary **cross-entropy loss** function với optimizer là **Adam** và metric là **accuracy**.
- Tất cả ý tưởng trên được cấu hình trong class **SentimentCNN1D** trong file **deep_model.py**.

- Đọc dữ liệu train và test.

	raw_comment	normalize_comment	emoji_decode	label
0	form k đẹp lắm	form không đẹp lắm		0 0
1	Áo Rộng thật sự\nGi nhanh\nChất lượ...	áo rộng thật sự nhanh chất lượng v...		1 1
2	Màu túi hơi tối do với ảnh chụp	màu túi hơi tối do ảnh chụp		2 0
3	Chất liệu vải k ổn lắm	chất liệu vải không ổn lắm		3 0
4	Mã "màu đen" hiện lên áo màu đen, còn ...	mã màu đen hiện lên áo màu đen còn mã...		4 0
...
11359	Khi bán combo ghi 3 đôi, nhưng nhận chỉ ...	bán combo ghi đôi nhưng nhận chỉ đôi ta...		11359 0
11360	Áo đẹp, from chuẩn. Sẽ ủng hộ tiếp ạ !	áo đẹp from chuẩn sẽ ủng hộ tiếp		11360 1
11361	Đặt màu trắng kem thì giao màu xanh đen...	đặt màu trắng kem giao màu xanh đen giao... expressionless		11361 0
11362	Đồ chơi tí hon.\nShop nên dùng bán sp ...	đồ chơi tí hon nên dùng bán sản phẩ...		11362 0
11363	Son đẹp lắm nha mọi ng, đáng đồng tiề...	son đẹp lắm mọi đáng đồng tiền bát m...		11363 1
11364 rows × 3 columns				
	raw_comment	normalize_comment	emoji_decode	label
0	Shop bán hàng kí quần đầy đủ. Chất l...	bán hàng kí quần đầy đủ chất lượn...		0 1
1	Đặt xl, áo ghi xl nhưng lại bé tí tẹo ...	đặt xl áo ghi xl nhưng bé tí tẹo ngang ...		1 0
2	Kèp rất rất đẹp săn được giá sale h...	kèp rất rất đẹp săn được giá sale h...		2 1
3	K nghĩ áo đẹp vậy đâu giao hàng nhanh k...	không nghĩ áo đẹp giao hàng nhanh không ...		3 1
4	Hôm nay mới nhận dc nhưng nhìn qua thâ...	hôm nay mới nhận được nhưng nhìn qua...		4 1
...
2837	Minh đặt 2 dây buộc tóc nhưng lại chỉ...	đặt dây buộc tóc nhưng chỉ nhận đượ...	star slightly_smiling_face slightly_smiling_fa...	2837 0
2838	Chất vải đẹp, có vẻ mát, giao hàng nha...	chất vải đẹp có vẻ mát giao hàng nhanh...		2838 1
2839	Dây nhin tạm. K đẹp. Mắc sai dây mình ...	dây nhin tạm không đẹp mắc sai dây pha...		2839 0
2840	Quần đẹp chất lượng ok nhưng chữ ru...	quần đẹp chất lượng ok nhưng chữ cu...		2840 1
2841	Ok hơi chật tí	ok hơi chật tí		2841 1
2842 rows × 3 columns				

- Load model **fast text** cho tiếng Việt để tiến hành chuyển đổi các word thành các vector **100 dimensions**.

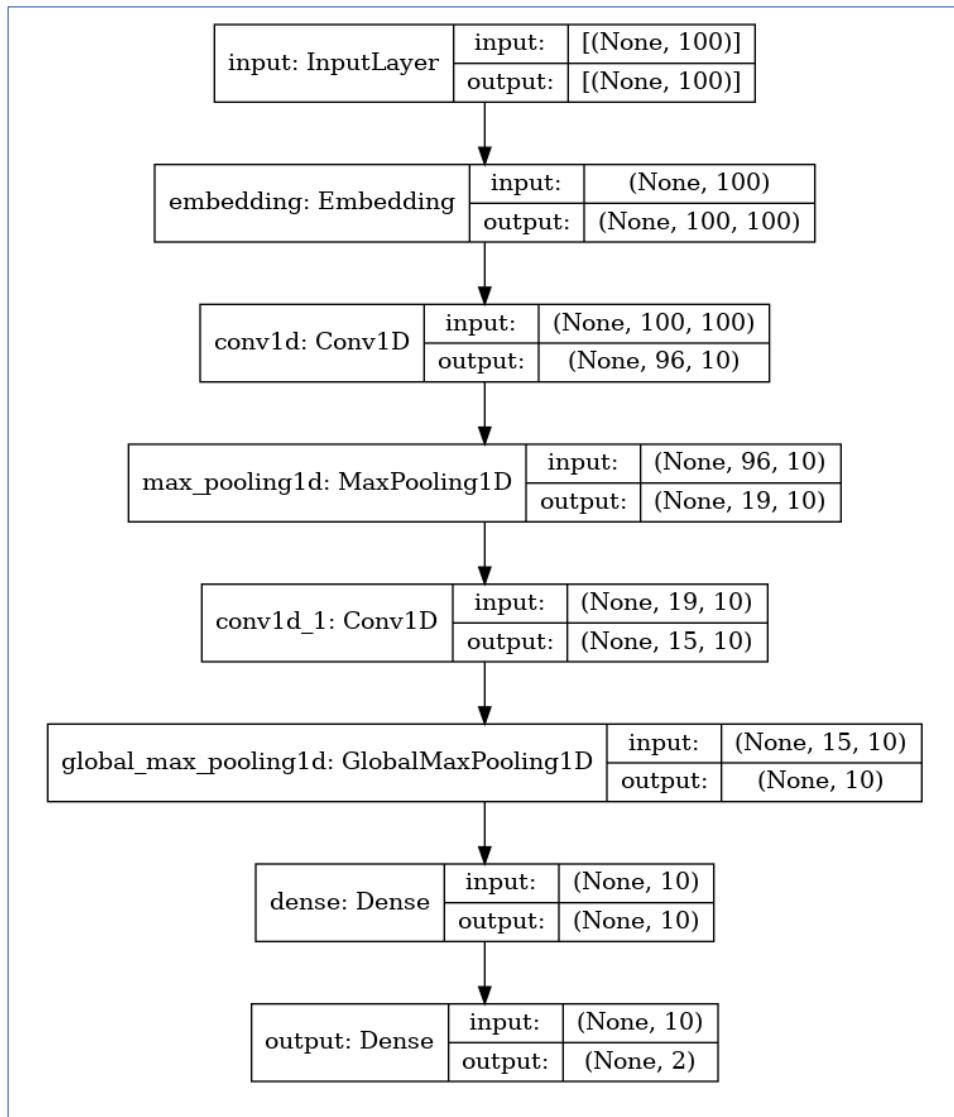
```
1 ft = fasttext.load_model("./cc.vi.100.bin")
```

Warning : `load_model` does not return WordVectorModel or SupervisedModel any more, but a `FastText` object which is very similar.

- Định nghĩa các Hyper-Parameters.

```
1 hyperparams_1 = {
2     'px': X_train['normalize_comment'],
3     'py': y_train,
4     'pfasttext': ft,
5     'pno_neurons': 10,
6     'pno_filters': 5,
7     'pnum_words': 3000,
8     'pseq_length': 100,
9     'pembedding_dim': 100,
10    'pbatch_size': 32,
11    'pepochs': 10,
12    'psave_path': './models/cnn1d_model_comments_1.h5'
13 }
14
15 model_1 = DeepModel.SentimentCNN1D()
16 model_1.define(**hyperparams_1)
17 model_1.model.summary()
18 plot_model(model_1.model, to_file='./images/cnn1d_model_comments_1.png', show_shapes=True, show_layer_names=True)
```

Model: "model"		
Layer (type)	Output Shape	Param #
input (InputLayer)	[None, 100]	0
embedding (Embedding)	(None, 100, 100)	358400
conv1d (Conv1D)	(None, 96, 10)	5010
max_pooling1d (MaxPooling1D)	(None, 19, 10)	0
conv1d_1 (Conv1D)	(None, 15, 10)	510
global_max_pooling1d (Global)	(None, 10)	0
dense (Dense)	(None, 10)	110
output (Dense)	(None, 2)	22
Total params: 364,052		
Trainable params: 5,652		
Non-trainable params: 358,400		



```

Epoch 1/10
320/320 [=====] - 2s 5ms/step - loss: 0.6710 - accuracy: 0.5945 - val_loss: 0.6354 - val_accuracy: 0.6482
Epoch 2/10
320/320 [=====] - 1s 4ms/step - loss: 0.6153 - accuracy: 0.6507 - val_loss: 0.6123 - val_accuracy: 0.6588
Epoch 3/10
320/320 [=====] - 1s 5ms/step - loss: 0.5955 - accuracy: 0.6630 - val_loss: 0.6060 - val_accuracy: 0.6588
Epoch 4/10
320/320 [=====] - 1s 4ms/step - loss: 0.5840 - accuracy: 0.6717 - val_loss: 0.6141 - val_accuracy: 0.6500
Epoch 5/10
320/320 [=====] - 1s 4ms/step - loss: 0.5757 - accuracy: 0.6723 - val_loss: 0.6043 - val_accuracy: 0.6544
Epoch 6/10
320/320 [=====] - 1s 4ms/step - loss: 0.5680 - accuracy: 0.6806 - val_loss: 0.6125 - val_accuracy: 0.6667
Epoch 7/10
320/320 [=====] - 1s 4ms/step - loss: 0.5617 - accuracy: 0.6859 - val_loss: 0.6097 - val_accuracy: 0.6570
Epoch 8/10
320/320 [=====] - 1s 4ms/step - loss: 0.5552 - accuracy: 0.6875 - val_loss: 0.6109 - val_accuracy: 0.6649
Epoch 9/10
320/320 [=====] - 1s 4ms/step - loss: 0.5493 - accuracy: 0.6928 - val_loss: 0.6246 - val_accuracy: 0.6544
Epoch 10/10
320/320 [=====] - 1s 4ms/step - loss: 0.5441 - accuracy: 0.6965 - val_loss: 0.6224 - val_accuracy: 0.6588

```

- **Nhận xét:**

- Có thể thấy, model này hoạt động không tốt so với LSTM, trong khi với số epoch tương ứng LSTM cho ra accuracy và loss value trên validation data tốt hơn nhiều.
- Như vậy, việc sử dụng các word vectors của fasttext không đem lại hiệu suất tốt cho tập dữ liệu của chúng ta, chúng ta có thể thay đổi kiến trúc của model để có kết quả tốt hơn, nhưng với hiện tại ta có thể phần nào khẳng định rằng các word vector này không đại diện tốt cho tập dữ liệu của chúng ta ⇒ Dừng phát triển model này.

3.3 Công việc 3. (File: 10.model.ipynb)

- Ở phần này chúng ta sẽ kết hợp **Emoji sentiment model** và **Comment sentiment model** lại thành một model duy nhất.
- Tiếp theo ta sẽ cho chúng dự đoán lại trên test data để xem khi kết hợp lại với nhau thì tổ hợp nào cho kết quả tốt nhất.
- Sau đó ta sẽ cho các tổ hợp model này dự đoán trên khoảng 10 comment mà ta sẽ lên trang chủ Shopee lấy về.
- Đọc dữ liệu test lên.

	raw_comment	normalize_comment	emoji_decode	label
0	Vãi ok . Mặc thoải mái. 60-65kg mac xxl ...	vãi ok mặc thoải mái kg mac xl la lan sau...	wink	0 1
1	Shop giao hàng nhanh chóng, đóng gói cẩn...	giao hàng nhanh chóng đóng gói cẩn thận...	heart	1 1
2	Minh mua cái áo len. Đã xác nhận nhận...	mua cái áo len xác nhận nhận hàng trê...	rofl rofl	2 0
3	áo đẹp, đúng size, form ok, kh biết có b...	áo đẹp đúng size form ok không biết có...	rofl rofl rofl rofl rofl rofl rofl rofl r...	3 0
4	Cặp xịn xò lém lun😍😍	cặp xịn xò lém luôn	heart_eyes heart_eyes	4 1
...
252	áo tạm được...Inquần thì như警示教育...	áo tạm được quần như	shit	252 0
253	Chả nhận được qq gì cả 😂. Sửa lại 3...	chả nhận được q gì cả sửa cho tự n...	joy	253 0
254	Áo nhau nátmặt ko dc.....😢😢😢😢😢	áo nhau nát mặt không được	disappointed disappointed disappointed disappo...	254 0
255	Rách 1 lỗ mà vẫn giao cho được 😢	rách lỗ mà vẫn giao cho được	cry	255 0
256	Áo đẹp, shop gói hàng cẩn thận, giao h...	áo đẹp gói hàng cẩn thận giao hàng nhanh	kissing_heart kissing_heart kissing_heart kiss...	256 1

- Nay giờ ta sẽ load tất cả các **Emoji sentiment model** và **Comment sentiment model** lên.
- Tiến hành ghép cặp từng **Emoji sentiment model** và **Comment sentiment model**. Ta có ba emoji sentiment model và 5 comment sentiment model, vậy tổng cộng ta có 15 model tổ hợp như vậy.
- Tiến hành kiểm tra một vài tổ hợp model trên test data để xem khi kết hợp hệ thống của chúng ta có chạy không.

	raw_review	probability	class
0	Vãi ok . Mặc thoải mái. 60-65kg mac xxl ...	(0.18665013249503054, 0.8133498806599009)	1
1	Shop giao hàng nhanh chóng, đóng gói cẩn...	(0.01988817364252765, 0.9801117991867729)	1
2	Mình mua cái áo len. Đã xác nhận nhận...	(0.6534688392760857, 0.3465311377828199)	0
3	áo đẹp, đúng size, form ok, kh biết có b...	(0.08410762570137753, 0.91589239762557)	1
4	Cặp xịn xò lém lun😍😍	(0.034524129719382346, 0.9654758712293343)	1
...
252	áo tạm được...\\nquần thì như 🍔 ...	(0.7727433524250488, 0.2272566634947465)	0
253	Chả nhận được qq gì cả 😂. Sửa lại 3...	(0.6193349746614896, 0.3806650485861045)	0
254	Áo nhau nátmặt ko đc.....😞😞😞😞😞	(0.880395587301454, 0.11960438575385607)	0
255	Rách 1 lỗ mà vẫn giao cho được 😢	(0.8353460421896457, 0.16465393272285242)	0
256	Áo đẹp, shop gói hàng cẩn thận, giao h...	(0.04836132829503487, 0.9516386656222646)	1

257 rows × 3 columns

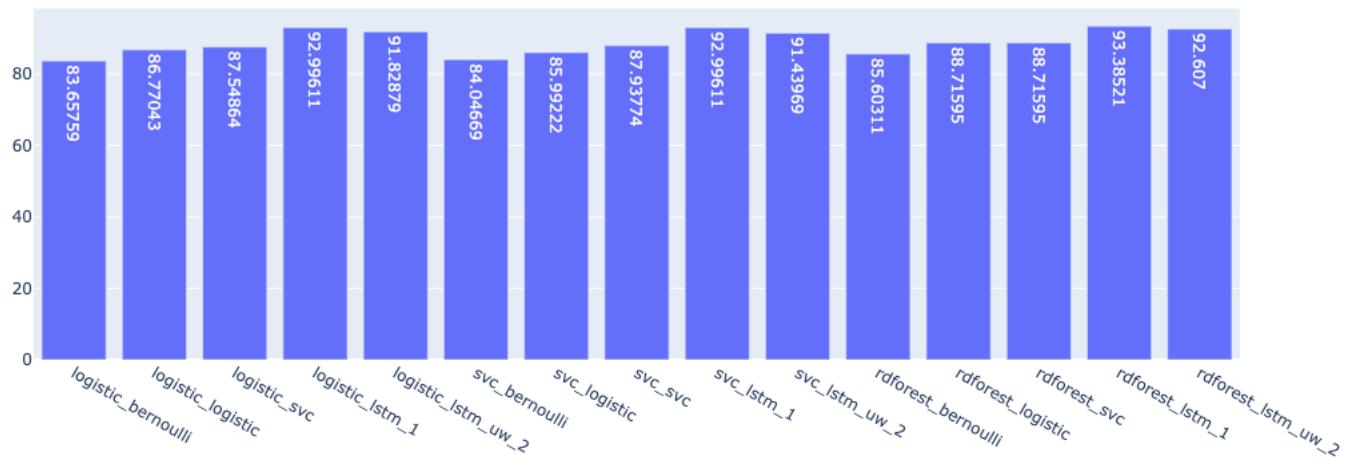
	raw_review	probability	class
0	Vãi ok . Mặc thoải mái. 60-65kg mac xxl ...	(0.6098593768522194, 0.3901406231477812)	0
1	Shop giao hàng nhanh chóng, đóng gói cẩn...	(0.03846174015554377, 0.9615382598444561)	1
2	Mình mua cái áo len. Đã xác nhận nhận...	(0.6535451811966995, 0.34645481880329865)	0
3	áo đẹp, đúng size, form ok, kh biết có b...	(0.19312646386724397, 0.8068735361327547)	1
4	Cặp xịn xò lém lun😍😍	(0.05537855658322438, 0.944621443416776)	1
...
252	áo tạm được...\\nquần thì như 🍔 ...	(0.7382678412307244, 0.26173215876927525)	0
253	Chả nhận được qq gì cả 😂. Sửa lại 3...	(0.6178695693822782, 0.3821304306177228)	0
254	Áo nhau nátmặt ko đc.....😞😞😞😞😞	(0.8391757036978025, 0.1608242963021982)	0
255	Rách 1 lỗ mà vẫn giao cho được 😢	(0.8332168638415237, 0.16678313615847448)	0
256	Áo đẹp, shop gói hàng cẩn thận, giao h...	(0.06536477199913324, 0.9346352280008661)	1

257 rows × 3 columns

- Ở đây câu hỏi đặt ra là làm sao chúng ta có thể kết hợp kết quả từ hai model?
- Giả sử nếu các model của chúng ta dự đoán ra kết quả là 0 và 1, thì với hai model ta sẽ gặp khó khăn trong trường hợp:

- Emoji sentiment model dự đoán 0 và Comment sentiment model dự đoán 1.
 - Emoji sentiment model dự đoán 1 và Comment sentiment model dự đoán 0.
- Khi rơi vào những trường hợp này, ta khó lòng chọn đâu là kết quả phù hợp. Nên ở đây thường người ta sẽ đào tạo 3 model và cho chúng voting. Với RandomForest nó cũng làm điều này nhưng sklearn sử dụng kĩ thuật là Bagging Voting.
 - Một cách mà nhóm dùng để xử lý vấn đề này là nhóm sẽ cho toàn bộ các model trả về giá trị xác suất, đó là lí do mà các model Deep Learning sử dụng activation là **softmax** thay vì **sigmoid**. Ở sklearn thì ngoại trừ các model SVC ta phải bật cờ **probability=True** thì toàn bộ các classifier khác đều hỗ trợ dự đoán xác suất, lúc này ta đơn giản lấy trung bình cộng giữa các class tương ứng giữa **Emoji sentiment model** và **Comment sentiment model** so sánh xem xác suất nào lớn hơn thì review này thuộc vào nhóm đó.
 - Trong trường hợp review chỉ có emoji hoặc comment, ta xử lý còn dễ dàng hơn là chỉ sử dụng model tương ứng để dự đoán.
 - Nay ta sẽ cho 15 tổ hợp model này dự đoán lại trên test data sau đó đánh giá accuracy.

```
[[ 'logistic_bernoulli' '0.8365758754863813' ]
 [ 'logistic_logistic' '0.867704280155642' ]
 [ 'logistic_svc' '0.8754863813229572' ]
 [ 'logistic_lstm_1' '0.9299610894941635' ]
 [ 'logistic_lstm_uw_2' '0.9182879377431906' ]
 [ 'svc_bernoulli' '0.8404669260700389' ]
 [ 'svc_logistic' '0.8599221789883269' ]
 [ 'svc_svc' '0.8793774319066148' ]
 [ 'svc_lstm_1' '0.9299610894941635' ]
 [ 'svc_lstm_uw_2' '0.914396887159533' ]
 [ 'rdforest_bernoulli' '0.8560311284046692' ]
 [ 'rdforest_logistic' '0.8871595330739299' ]
 [ 'rdforest_svc' '0.8871595330739299' ]
 [ 'rdforest_lstm_1' '0.933852140077821' ]
 [ 'rdforest_lstm_uw_2' '0.9260700389105059']]
```



- **Nhận xét:**

- Ở đây sau khi kết hợp lại thì có 13 model cho accuracy trên 85%.
 - 6 model cho accuracy hơn 90% và các model này đều có **comment sentiment model** sử dụng LSTM.
 - Một điều thú vị là ở những bước trước ta đánh giá **Emoji sentiment model** dùng RandomForest bị overfitting nhưng khi kết hợp với LSTM model thì nó cho ra accuracy cao nhất lên đến 93%. Rõ ràng một điều là ta không thể biết trước được điều gì hay khả năng chắc chắn được điều này trên dữ liệu của chúng ta.
- Bây giờ ta sẽ đi lấy 10 review ngẫu nhiên trên Shopee và cho các model dự đoán trên các review mới này. Năm review đầu tiên được lấy từ các review có 4 và 5 sao, 5 review cuối cùng được lấy từ các review có từ 3 sao trở xuống.

review	0	1	2	3	4	5	6	7	8	9
áo đẹp, bổ mình rất ưng và rất thic nhé. lên m...	Áo khá ấm, vải mềm, không có chí thura, tuy nhì...	Áo nỉ khá dày, nhẹ tênh, giống hình, thời gian...	Chất ồn nhưng dày quần như bị ố nhìn khá xấu. ...	Mỏng, chất dễ xù, ban đầu thấy chất khá cứng, ...	Giao màu khác trong ánh. Nhìn già kinh lươn. C...	Lười chằng buồn chụp cái áo vì thất vọng toàn ...	Áo màu k giống ánh phản viền là màu nâu đậm th...	Áo khác trên ảnh nhiều lắm màu nhạt dã man xo...	Áo rộng thùng thình bổ mình mặc còn vữa nên mi...	
logistic_bernoulli	1	0	1	0	0	0	0	0	0	0
logistic_logistic	1	0	1	1	0	0	0	0	0	0
logistic_svc	1	0	1	1	0	0	0	0	0	1
logistic_lstm_1	1	1	1	1	1	0	0	0	0	0
logistic_lstm_uw_2	1	0	1	1	0	0	0	0	0	0
svc_bernoulli	1	0	1	0	0	0	0	0	0	0
svc_logistic	1	0	1	1	0	0	0	0	0	0
svc_svc	1	0	1	1	0	0	0	0	0	1
svc_lstm_1	1	1	1	1	1	0	0	0	0	0
svc_lstm_uw_2	1	0	1	1	0	0	0	0	0	0
rdforest_bernoulli	1	0	1	0	0	0	0	0	0	0
rdforest_logistic	1	0	1	1	0	0	0	0	0	0
rdforest_svc	1	0	1	1	0	0	0	0	0	1
rdforest_lstm_1	1	1	1	1	1	0	0	0	0	0
rdforest_lstm_uw_2	1	0	1	1	0	0	0	0	0	0

- **Nhận xét**

- Có thể thấy toàn bộ các model có **Comment sentiment model** sử dụng **lstm_1** đều cho ra kết quả chính xác. Các model sử dụng các traditional classifier ở **Comment sentiment model** dự đoán còn sai nhiều ở các comment positive nhưng lại nhầm sang negative.
- Từ đây, có thể cho ra kết quả như sau, đối với Shopee review dataset của chúng ta, nếu xét riêng về phần comment thì các model Deep Learning hoạt động tốt hơn hẳn, điều này có lẽ vì do việc cấu trúc của comment ảnh hưởng lớn đến kết quả phân lớp.
- Trong 15 model, các model có phần **comment sentiment model** sử dụng **lstm_1** cho ra kết quả tốt, ta có thể giữ lại và cải thiện hoặc phát triển về sau.

4. Đánh giá

STT	Tiêu chí	Hoàn thành (%)
1	Đặt ra các vấn đề cần giải quyết.	100
2	Mô tả dữ liệu liên quan.	100
3	Chọn lựa, giải thích tính phù hợp của các mô hình học máy trên dữ liệu và bài toán nêu ra.	100
4	Thực hiện huấn luyện và mô tả chi tiết các thuật toán được triển khai để huấn luyện.	100
5	Mô tả cách phân chia dữ liệu huấn luyện và kiểm thử, lý giải cách phân chia và chứng tỏ kết quả không quá phụ thuộc vào cách phân chia đó.	100
6	Giải thích các độ đo để đánh giá mô hình.	100
7	Phân tích và trực quan hóa kết quả thu được, lý giải các điểm quan trọng.	100
8	Kết luận về vấn đề nêu ra ban đầu.	100

5. Tài liệu tham khảo

- [Hands-On Deep Learning Algorithms with Python]
(<https://www.packtpub.com/product/hands-on-deep-learning-algorithms-with-python/9781789344158>)
- [Keras Deep Learning Cookbook]
(<https://www.amazon.com/Keras-Deep-Learning-Cookbook-implementing/dp/1788621751>)
- [Applied Text Analysis with Python]
(<https://www.oreilly.com/library/view/applied-text-analysis/9781491963036>)
- [Deep Learning Quick Reference]
(<https://www.packtpub.com/product/deep-learning-quick-reference/9781788837996>)
- [Advanced Natural Language Processing with TensorFlow 2]
(<https://www.amazon.com/Advanced-Natural-Language-Processing-TensorFlow/dp/1800200935>)
- [Natural Language Processing with TensorFlow]
(<https://www.packtpub.com/product/natural-language-processing-with-tensorflow/9781788478311>)
- Và một số tài liệu, bài viết trên các trang GeekForGeeks, Medium, TowardsDataScience, TensorFlow,...

6. Mã nguồn

- Link Google Drive:
<https://drive.google.com/drive/folders/1ugt27sn6dQiBYJjXUDcRpTWEOfCjvMfq?usp=sharing>