Report

109550005 張可晴

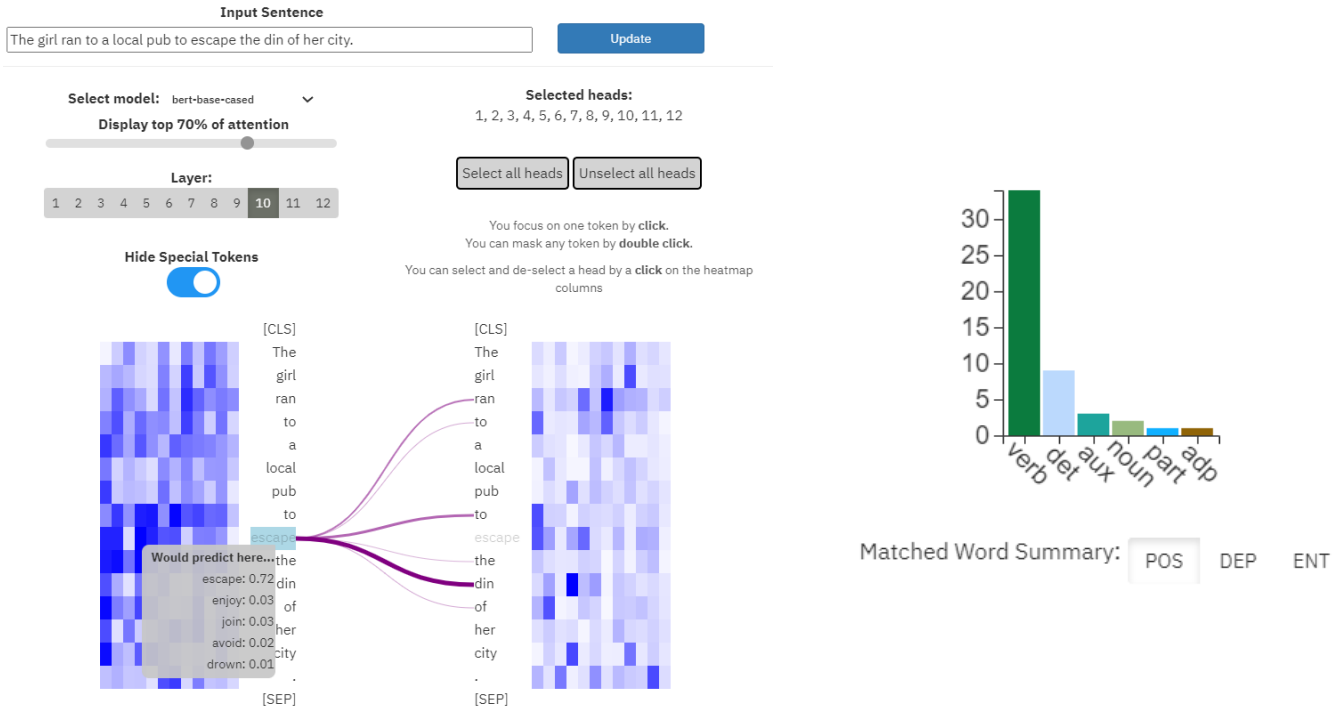
Describe your understanding and findings about the attention mechanism by exBERT

As the paper mentions that neural networks are based on the Transformer architecture, such as BERT. And the transformer is based on subsequent application of “multi-head attention” to route the model reasoning. By using exBERT, we can have both the attention and the token embeddings knowledge for the user-defined model and corpus by probing whether the representations capture metadata such as linguistic features or positional information.

[bert-base-cased]

Behind the mask

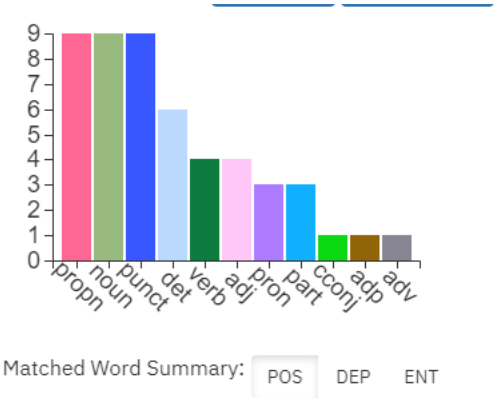
With the technique called Masked Language Modeling (MLM), replacing the specific token with [MASK] to train the model, BERT needs to learn linguistic features in context by attention mechanism, as the model has no idea what the masked token is



From the above figure, I masked “escape” in the sentence: “The girl ran to a local pub to escape the din of her city.” I observe that BERT can precisely find the token “ran”, “to”, and “din” related to it at layer 10, the matched word summary are verb, and it can predict the masked word to be “escape” by 70% chance, which can

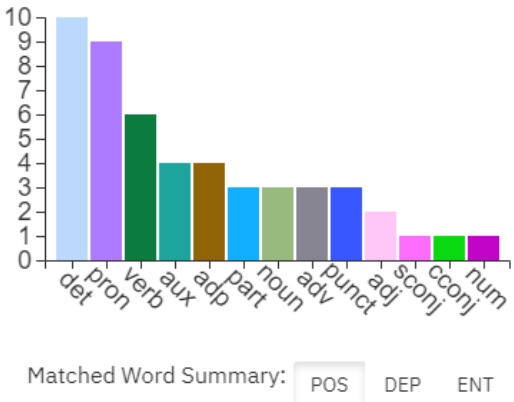
prove the attention mechanism applied in BERT.

Behind the heads



As for this instance, I use the sentence: “Cassie ran to a local pub. She can finally feel relaxed.”, masking “She”. In layer 12, we can find that BERT can find that the matched word is highly possible to be pronoun or noun. Moreover, it will focus on “Cassie” and predict the masked word would be “I” or “She” from the context, which indicates that BERT can pay attention on the last sentence and determine that “Cassie” is a girl’s name.

More than position

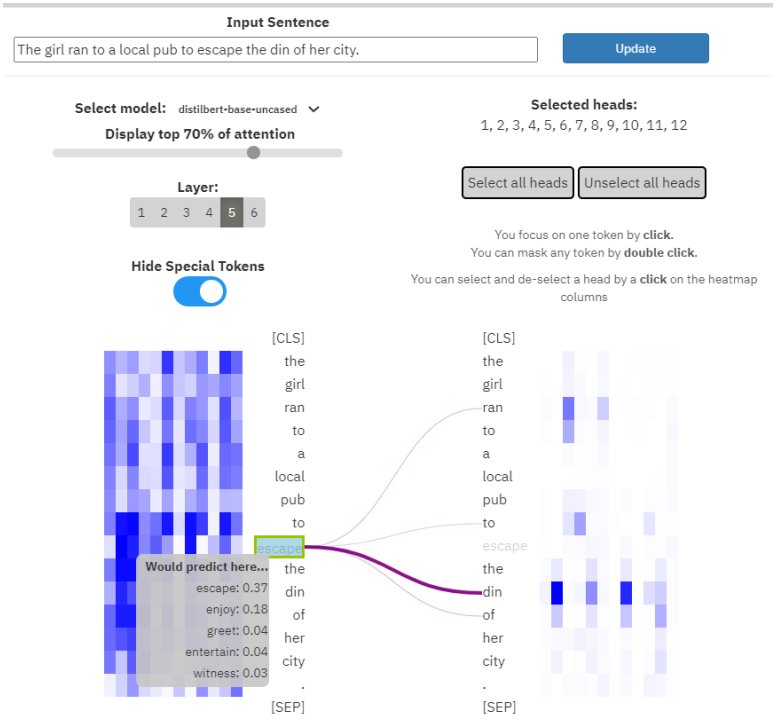


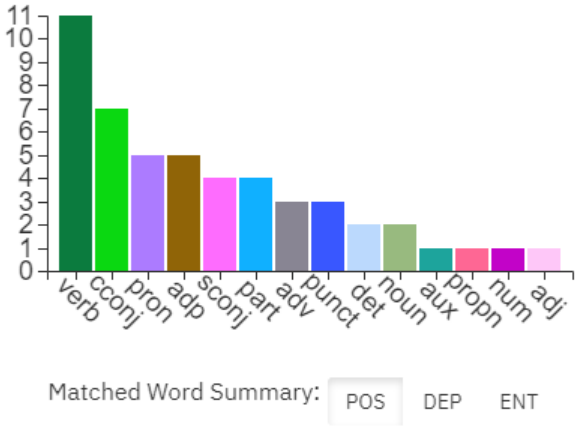
In this example, I wanted to observe whether BERT can know that certain token needs to pay attention on preceding token or succeeding one. As a result, I used the sentence: “Cassie ran to a local pub and met her classmate whose name is Jake.” and masked “her” at the same time. From the above figure, we can find that BERT won’t determine that the masked token is related to “Jake”, which is wrong. Instead, BERT can successfully predict that the masked token has a relation with “Cassie”, predicting it is likely to be “her” in layer 11, and most match word is det. To sum up, we can confirm that BERT has the ability to judge if it needs to focus on preceding token or succeeding one.

[distilbert-base-uncased]

DistilBERT is a distil version of BERT that it can achieve 97% of the original BERT’s language understanding capability and even be equipped with 60% faster operation ability with 40% less size. Next, I will perform 3 observations like the last part.

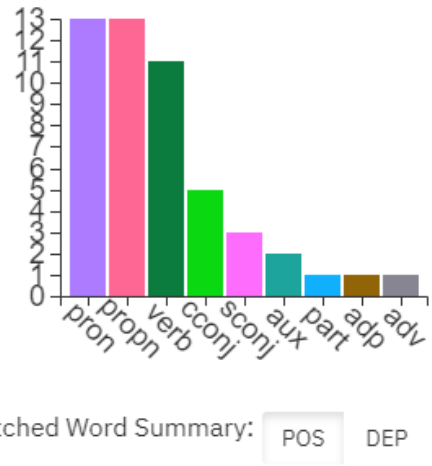
Behind the mask





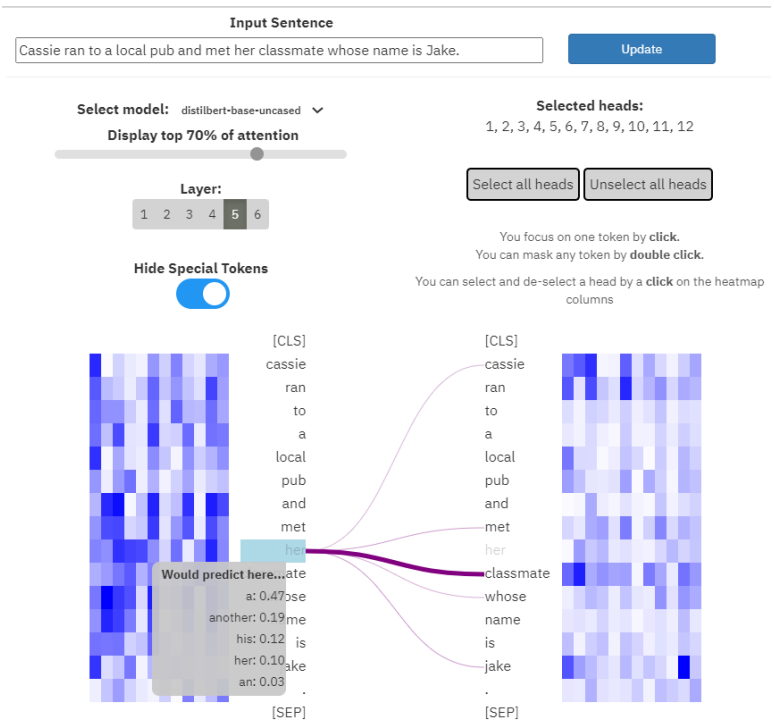
In this part, I used the sentence “The girl ran to a local pub to escape the din of her city.” and masked “escape” simultaneously. The result is very similar to BERT’s that it can also find that “ran”, “to”, and “din” have a relation with the masked token. What’s better, it can correctly predict the masked token to be “escape”, but with 37% possibility, which is much lower than BERT’s. From the mentioned above, we have confidence to ensure that distilBERT also has the ability to perform self-attention, while it may have a little worse performance than BERT.

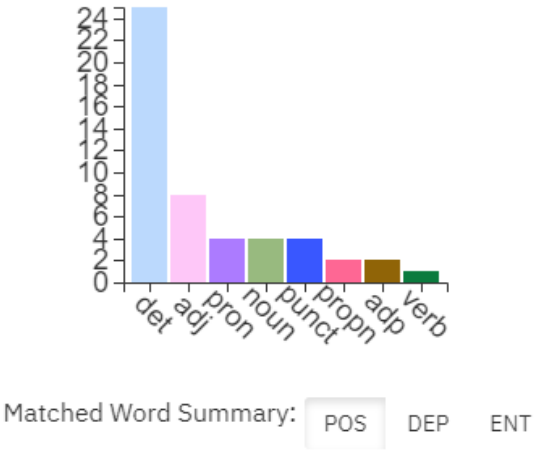
Behind the heads



The same with the last part, with sentence ”Cassie ran to a local pub. She can finally feel relaxed” and masking “she”, distilBERT can predict the masked token to be “Cassie” or “she”, revealing that it can understand the context owning to the self-attention mechanism. Additionally, in the light of this example, it seems that distilBERT performs as well as BERT.

More than position





In this part, I masked “her” in “Cassie ran to a local pub and met her classmate whose name is Jake.” it turn out to be a little misunderstand the context since it predict the masked token is “her” with only 10% possibility, while “his” has higher posiobility.

In conclusion, we can find both BERT and distilBERT equipped with attetion mechanism through exBERT, while BERT will obviously outperform distilBERT when the sentence is hard to understand.

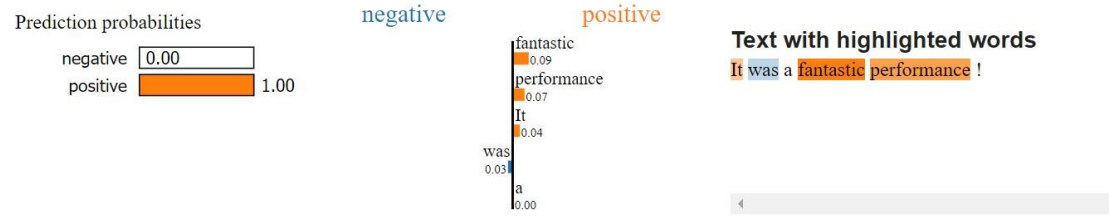
Compare at least 2 sentiment classification models

In this part, I firstly choose two sentences:

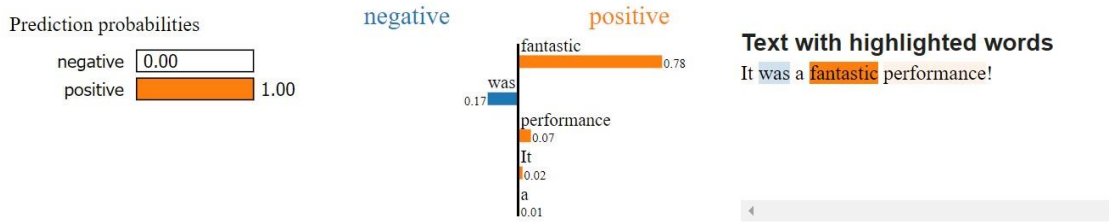
“It was a fantastic performance!”

“That is a terrible movie.”

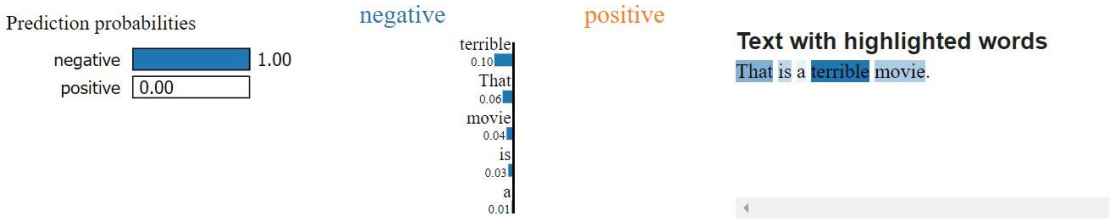
And I will compare two sentiment classification models given by TA by LIME.



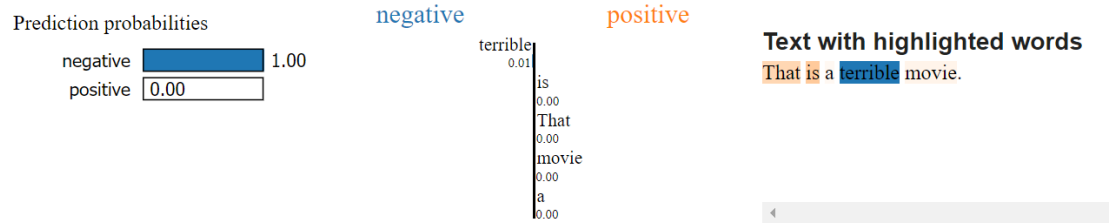
Model 1(distilBERT: It was a fantastic performance!)



Model 2(smallBERT: It was a fantastic performance!)



Model 1(distilBERT: That is a terrible movie.)

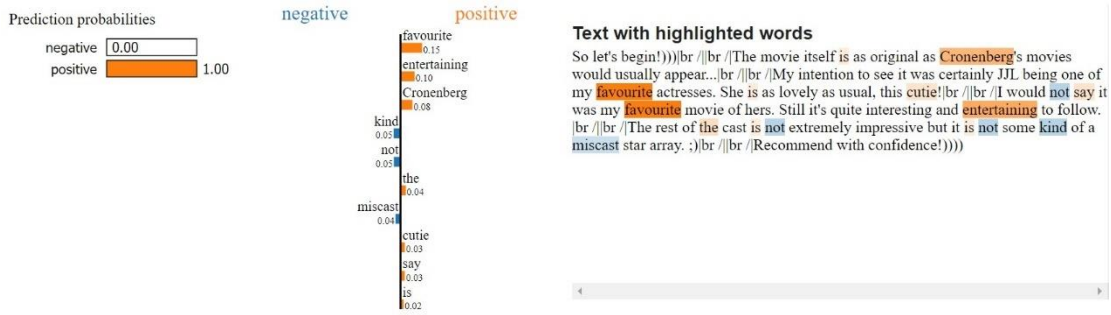


Model 2(smallBERT: That is a terrible movie.)

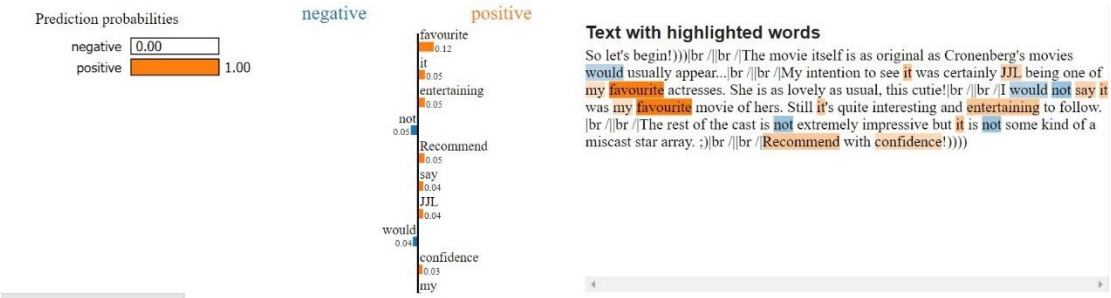
From the figure above, we can find that both distilBERT and smallBERT can

successfully distinguish if the sentence is positive or negative, while they are quite simple sentences, so there's no distinct difference between these two models.

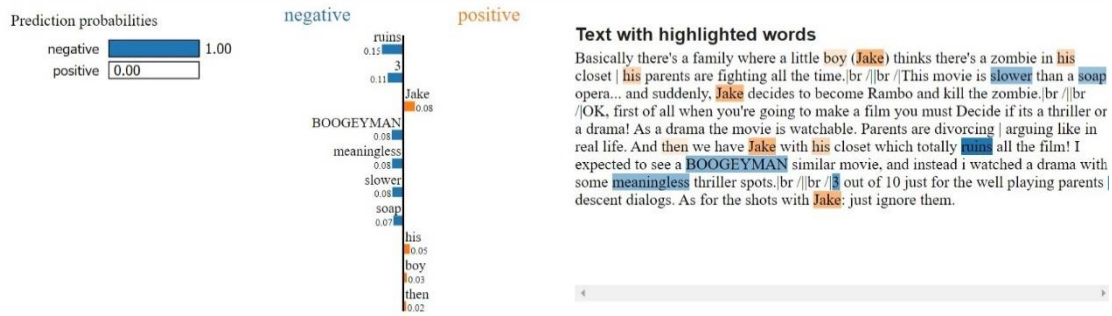
Moreover, I chose three movie reviews from IMDB. Two of them are positive and one of them is negative.



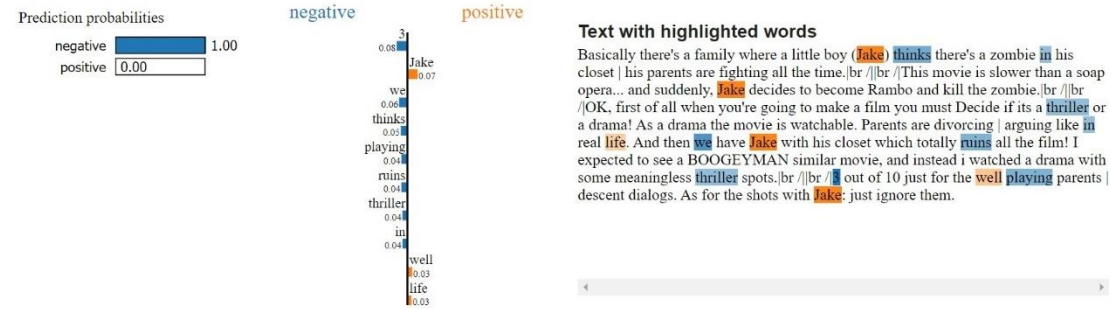
Example 1: Model 1(positive)



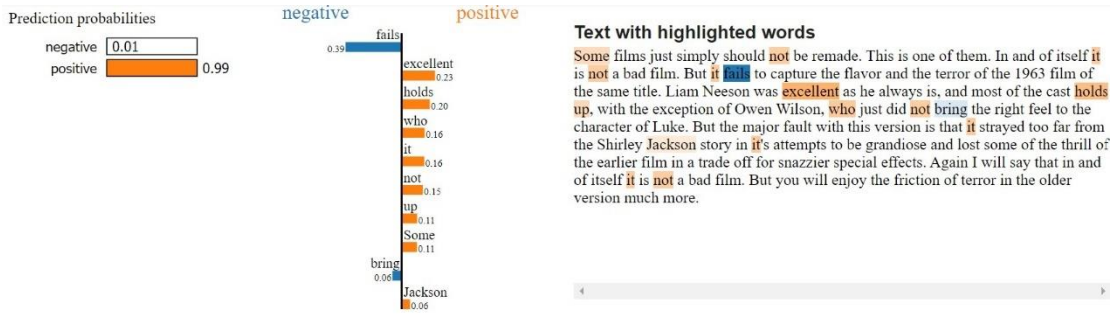
Example 1: Model 2(positive)



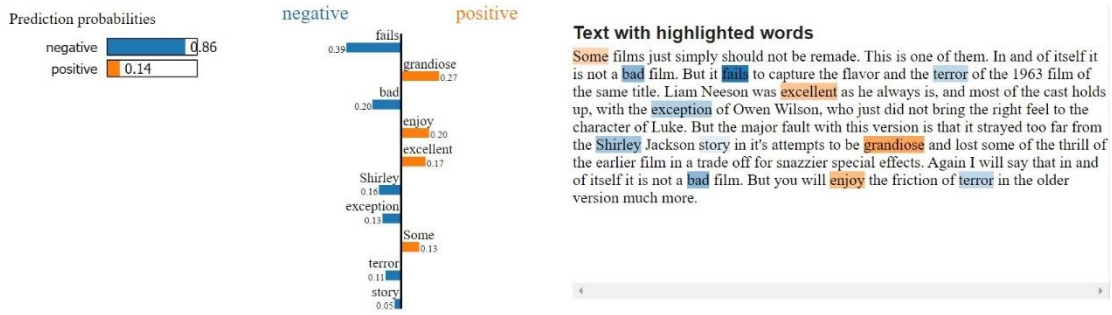
Example 2: Model 1(negative)



Example 2: Model 2(negative)



Example 3: Model 1(positive)



Example3: Model 2(positive)

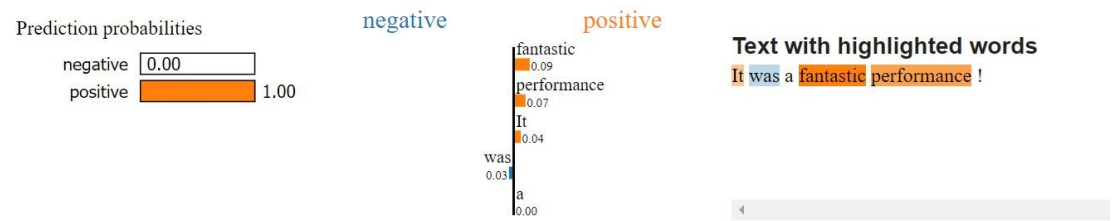
Comparing the results from the two models, I found that the first model would outperform the second one. As the second model need to judge the more confused review, it will misunderstand the meaning of the sentence, while the first one can understand what it means. Let's take a look on example 3. We can observe that model 1 can successfully tell that this review is 99% positive. As for model 2, it not only fails to judge the review is positive, but also tell us that the review would be 86% negative. In my humble opinion, the reason why model 1 will perform better than the model 2 is because that the latter applys prajjwal1/bert-small to tokenize and pretrain the model. prajjwal1/bert-small obtained from converting Tensorflow checkpoint found in official Google BERT repository, which is pretrained by other data leading to lower accuracy. As a result, the pure distilBERT (model 1) will outperform smallBERT (model 2) on those reviews from IMDB, which can apparently be observed in example 3.

Compare the explanation of LIME and SHAP.

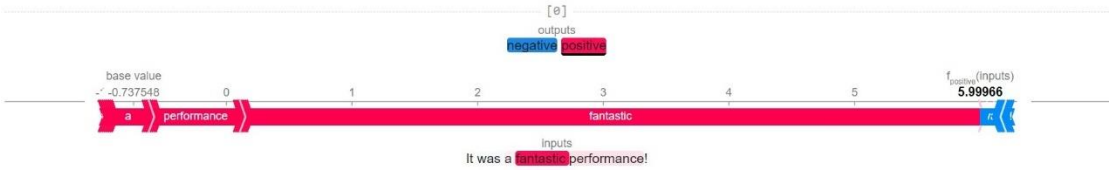
I will firstly apply pure distilBERT to do the comparison between LIME and SHAP. As the last part, I will initially use simple sentences:

"It was a fantastic performance!"

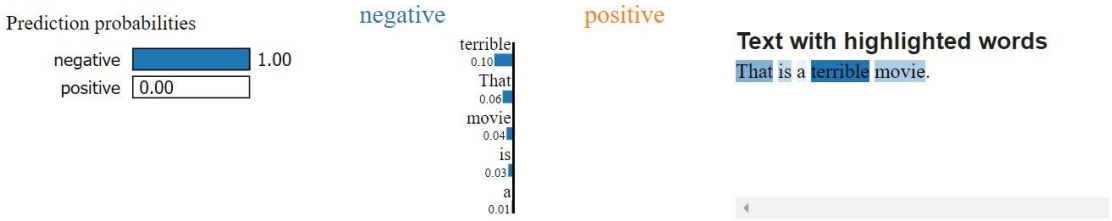
"That is a terrible movie."



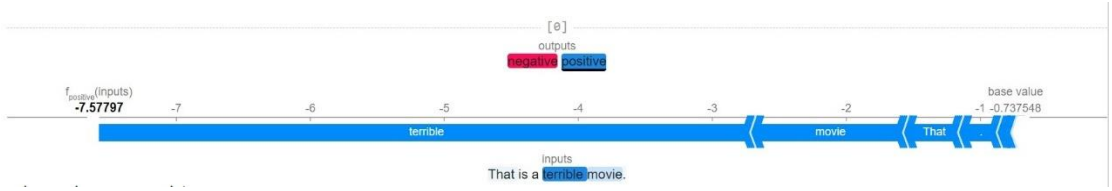
LIME (It was a fantastic performance!)



SHAP (It was a fantastic performance!)



LIME (That is a terrible movie.)



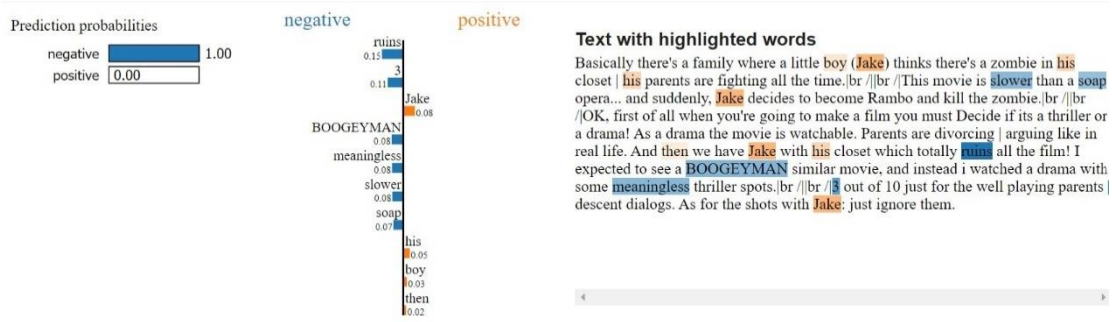
SHAP (That is a terrible movie.)

From the above figures, we can observe that there is a little difference between LIME and SHAP. Nevertheless, Both methods can give us a reasonable explanation for these simple and short sentences on the whole.

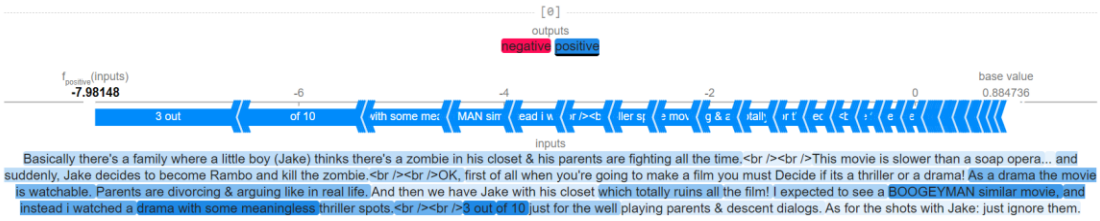
In addition, I compare LIME and SHAP with the reviews from IMDB.

LIME - Example 1: Model 1(positive)

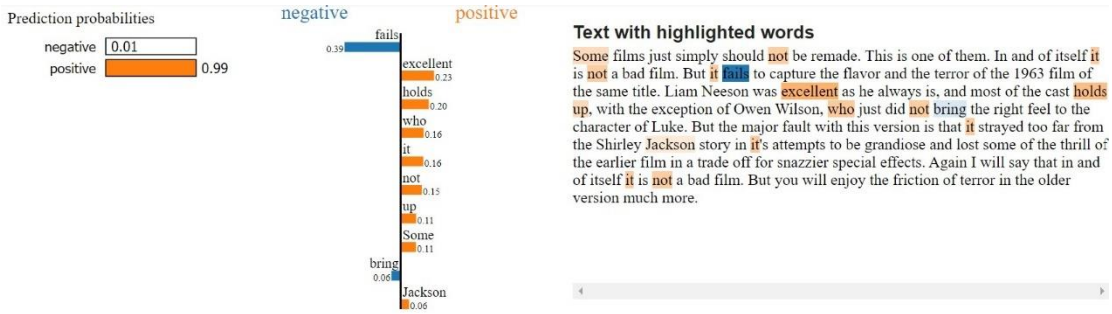
SHAP - Example 1: Model 1(positive)



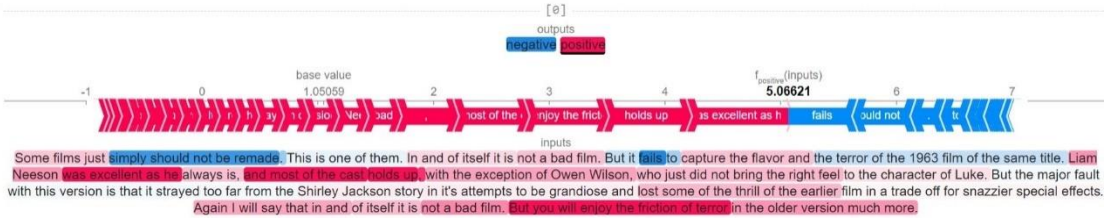
LIME - Example 2: Model 1(negative)



SHAP - Example 2: Model 1(negative)



LIME - Example 3: Model 1(positive)



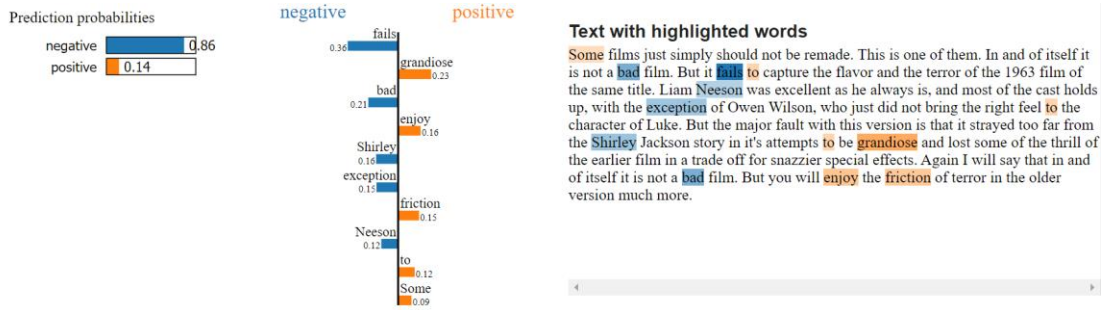
SHAP - Example 3: Model 1(positive)

As for the multiple sentences like these reviews from IMDB, we can observe that SHAP can explain the whole sentence at one time, while LIME can only give us the explanation of some specific and single word each time, leading to some errors. In example 3, which is a positive review, we can observe that “not” contribute 15% positive prediction, while the whole sentence is “Some films just simply should not be remade.”, which is apparently a negative comment. As for SHAP, it can explain the whole sentence and predict “simply should not be remade” as a negative comment that give the second important contribution to negative comment. Thus, I believe that SHAP can give us better explanation than LIME.

There are two reasons why SHAP outperforms LIME when it comes to multiple

sentences. On the one hand, as the mentioned above, LIME can only perceived one word as feature, while SHAP can select variable number of words to be a feature, which can greatly improve its ability of explanation when faced with multiple sentences that understand the meaning of paragraph by sentence will be beneficial to its effectiveness.

On the other hands, because SHAP has three properties: local accuracy, missingness, and consistency that LIME doesn't, enabling SHAP to be more consistent with human intuition, thus performing better than LIME. To prove above statement, I get another explanation of example 3 from LIME, using smallBERT as model.



In the figure above, we can ensure that LIME can't satisfy local accuracy as prediction value will change when the model is different. For example, "bad" account for 21% in negative comment for model 2, while it doesn't account for negative comment at all for model 1.

Last but not least, I use smallBERT to observe the difference between LIME and SHAP. Moreover, I especially pay attention on example 3 the most confused review.

LIME - Example 3: Model 2(positive)

SHAP - Example 3: Model 2(positive)

It turns out that SHAP also gives the wrong prediction. Nonetheless, we can find

that SHAP can successfully tell the if specific sentence is positive or negative. For instance, we can observe the first sentence. LIME will misunderstand the first word “Some” to be positive word. In the light of SHAP, it can explain the sentence “Some films just simply should not be remade.” to be negative comment, which is the right explanation.

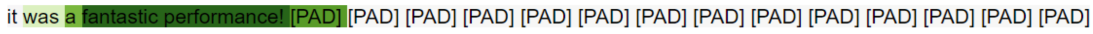
Describe how you implement other explanation techniques. And discuss with the explanation result

[integrated gradients]

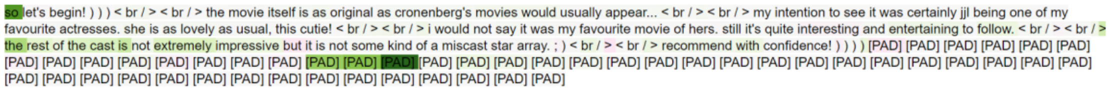
```
# We need only the input ids for the classification
x_test_sample = z_test_sample['input_ids']

# Preparation of Attention Masks
kwargs = {k: tf.constant(v) for k,v in z_test_sample.items() if k ==
'attention_mask'}
#   Extracting the first transformer block
b1 = auto_model1.layers[0].layers[0].transformer.layer[1]
n_steps = 20
method = "gausslegendre"
internal_batch_size = 5
ig    = IntegratedGradients(auto_model1,
                                                layer=b1,
                                                n_steps=n_steps,
                                                method=method,
                                                internal_batch_size=internal_batch_size)
predictions = auto_model1(x_test_sample, **kwargs).numpy().argmax(axis=1)
explanation = ig.explain(x_test_sample,
                                               forward_kwargs=kwargs,
                                               baselines=None,
                                               target=predictions)
attrs = explanation.attributions[0]
attrs = attrs.sum(axis=2)
from IPython.display import HTML
# Return HTML markup which highlights the text with a desired color.
def   hlstr(string, color='white'):
       return f"<mark style=background-color:{color}>{string} </mark>"

# Calculates color based on attribution values
def colorize(attrs, cmap='PiYG'):
       cmap_bound = np.abs(attrs).max()
       norm = mpl.colors.Normalize(vmin=-cmap_bound, vmax=cmap_bound)
       cmap = mpl.cm.get_cmap(cmap)
```

```
       colors = list(map(lambda x: mpl.colors.rgb2hex(cmap(norm(x))), attrs))
       return colors
words = tokenizer.decode(x_test_sample[0]).split()
colors = colorize(attrs[0])
HTML("".join(list(map(hlstr, words, colors))))
```

The above screenshot is the main part about how to apply integrated gradients to do sentimental analysis. Integrated gradients initially use baseline which is only the start token and end token without any emotional word. As the step increase, it adds up the sentence by each token. In this way, it can calculate each token’s contribution to the positive or negative meaning to the sentence. Following are some examples used in Integrated gradients.



It was a fantastic performance!



IMDB review (example1)

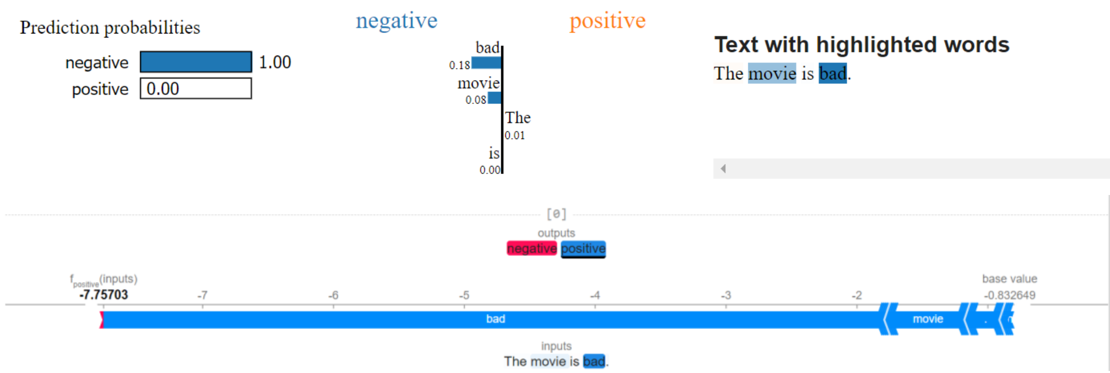
As for the first one, we can find that integrated gradients predicts that "fantastic" and "performance" give the same contribution to the positive meanings, which is even worse than LIME, not to mention SHAP. As for the longer sentences, it can successfully explain this paragraph is positive overall, while it thinks that "So" plays the most important part to make the sentence positive meaning, which is incorrect and perform worse than LIME and SHAP.

Try 3 different input sentences for attacks. Also, describe your findings and how to prevent the attack if you retrain the model in the future.

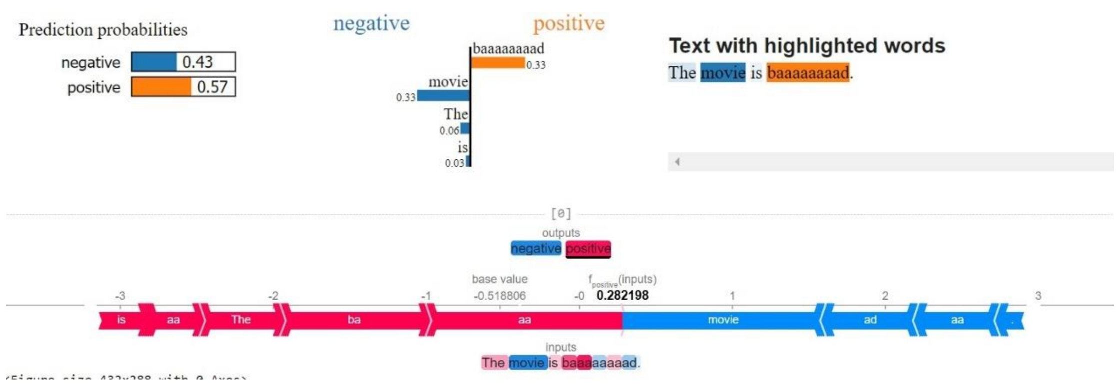
In this part, I will apply pure distilBERT to perform some attack.

[misspelling]

The original sentence: "The movie is bad."



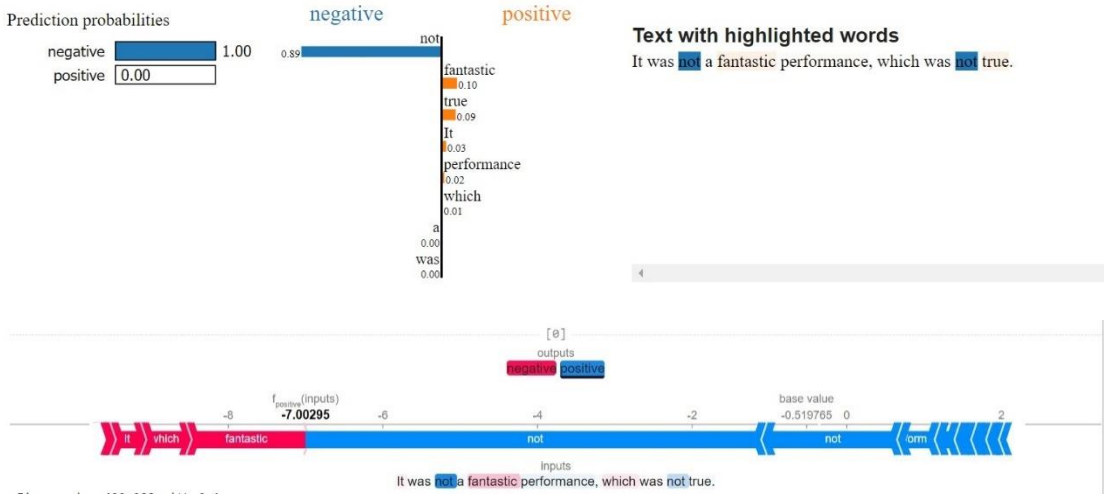
Both LIME and SHAP can explain precisely that this sentence is 100% negative meaning mainly due to the "bad". Then, I try to launch an attack by misspelling "bad" with "baaaaaaaad".



There is 57% confidence that the sentence has positive meaning, which implies that the misspelling attack is successful.

[double negation]

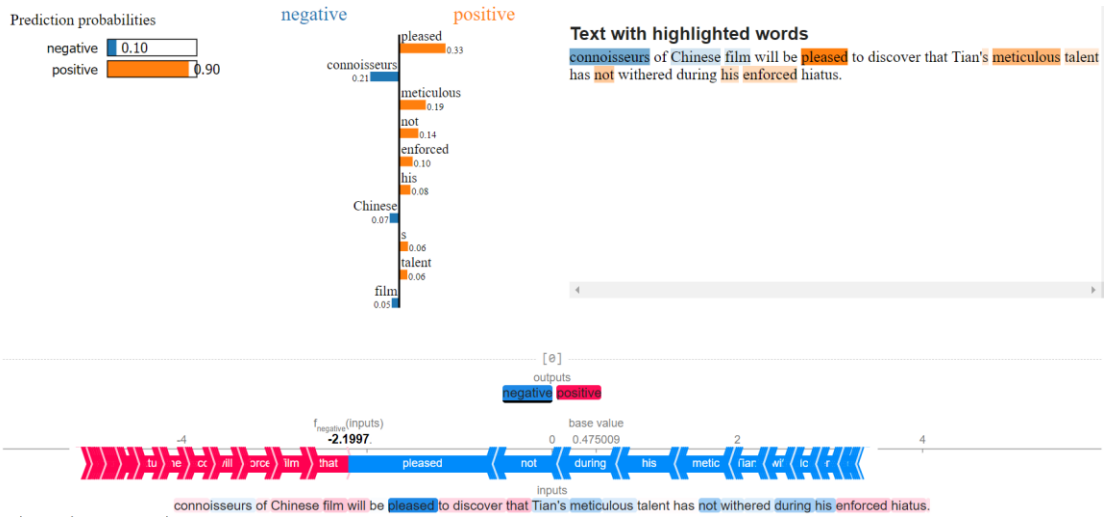
I launch an attack by using the double negative sentence “It was a fantastic performance, which was not true.” to confuse the explainer.



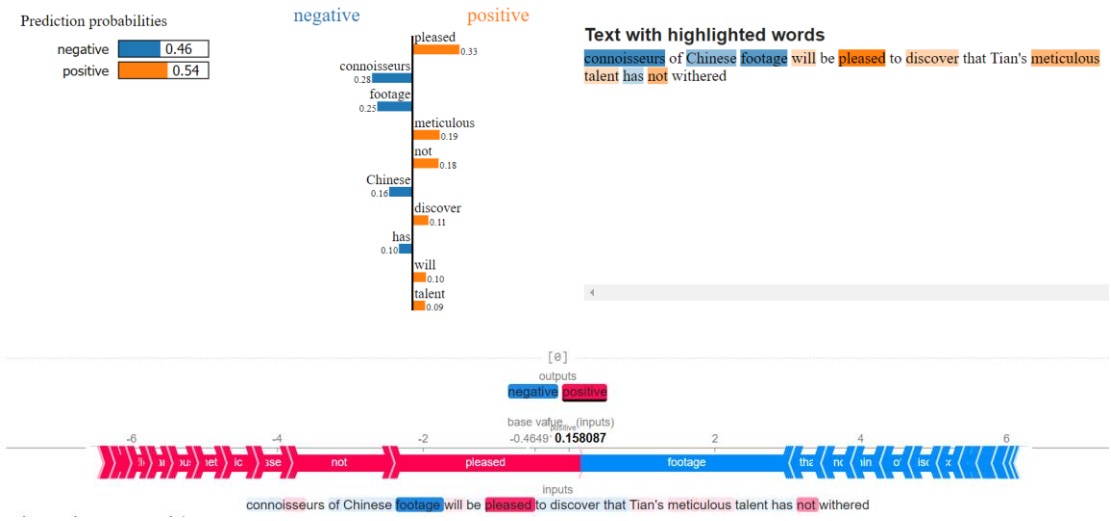
Both explainers are confused by the sentence that they treat the sentence as negative one owning to the occurrence of “not”. LIME even 100% believe that the sentence has negative meaning, indicating a totally successful attack.

[Word substitution by synonym]

I found a sentence on the Internet: “connoisseurs of Chinese film will be pleased to discover that Tian’s meticulous talent has not withered.”



SHAP has 90% confidence that it is positive sentence. Then, I try to attack by substituting the “film” for “footage”.



the explainer then predicts the sentence to be 54% chance to positive comment. It reveals that I attack successfully by substituting the synonym.

[prevention]

To prevent such attack, I can put more data with double negative sentences, misspelling ones, word substituting by synonym when I pretrain the model. Moreover, I can randomly add or drop some word during pretaining the model to resist other type attack as well.

Describe problems you meet and how you solve them

I believe that in this homework, the most challenging part is to find the useful sentence to test our model and explainer or we can't observe the situation we want. For example, I wanted to find the semantic similar word to give a successful attack, while I gave lots of tries but fail to find the useful sentence. Eventually, I had no choice but use the example on the Internet.