Home Task Report
Grasp

Task 1: Paraphrase and Semantic Textual Similarity

1.1 Road Map:

Initial Thoughts: this paper was created in 2015, which is prior to the inception of attention and transformer. My first initial thought upon parsing the README.md and the original SemEval2015 repository was to use Sentence Transformer. I am very familiar with the HuggingFace Sentence Transformer framework due to my previous research. I already have a very good idea of how to implement a Sentence Transformer pipeline to classify sequences. With this task, we have 2 sentences; thus we seek to compare the textual similarity between our sentences. The main goal can be simplified into a multi-label classification; where we classify the pair of sentences into 5 labels: 0, 1, 2, 3, 4, and 5 (with 0 being not similar to 5 being most similar).

First Approach: With that being said, due to my familiarity with sequence classification using Sentence Transformer, my first approach had the following inputs and outputs:

Input: Sentence_1 + [SEP] + Sentence_2
Output: [0, 0, 0, 1, 0]

I simply added a special [SEP] token to let the model know that our inputs consisted of 2 separate sentences. Moreover, as discussed, the output is multiple label; where the index of the 1 represent our label (0 to 5). With this initial approach, I could quickly implement the whole pipeline without much research. For evaluation metrics, I implemented a custom *compute_metrics* function, where I calculated: *f1_micro, accuracy,* and *recall*. I believe this process is essential toward a fast pace startup environment; where quick prototyping is vital. However, the performance was abysmal, where almost all metrics were close to 0 (~ 0.02 to 0.03). I hypothesize that this is due to the nature of multi-label classification and no information regarding the relationship between the input and output. In other words, the model could not understand that the output represent the similarity between the sentence 1 and sentence 2. Overall, I had 2–3 hours on Friday where I spent working/training on this approach. As soon as I realized that it is not a feasible method, I swiftly moved on.

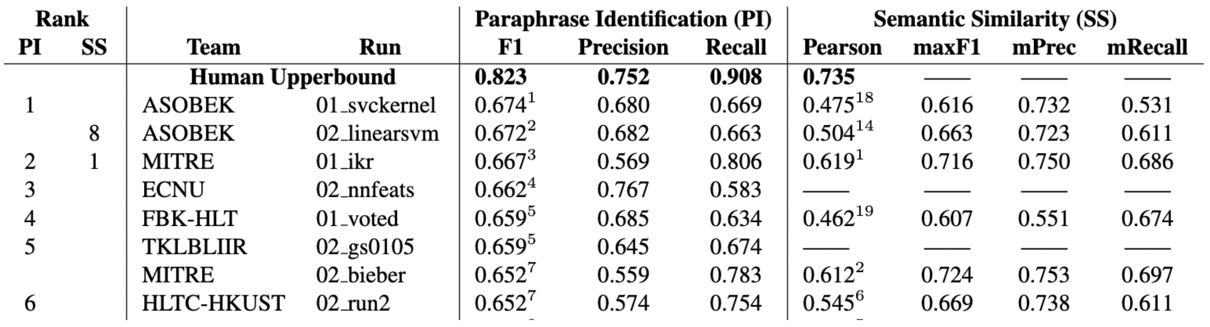
Final Approach: I had a class on recommender system where I learned about cosine similarity (where you can use this to find content "close" to the user's favorite content; thus recommending this "close" content. This gave me the initial idea for the second approach; where we have embeddings for our sentences, then calculate the cosine similarity scores between these embeddings. This link was very helpful in initially experimenting with this approach.

With that being said, the idea is the following:

1) normalize the similarity scores (from **0-5** to **0-1**)
2) pass the sentence 1 and 2 into our networks and generate the respective sentence embedding
3) the cosine similarity scores of the initial embeddings is compared to the gold similarity; as calculated in step 1. This is how the model will learn to generate embeddings such that the similarly will be as close as possible to the golden similarity scores.

With this pipeline, I experimented with *RoBERTa-base* and *all-MiniLM-L6-v2*. Both of which were better performing/comparable with the author's results without any fine-tuning. This is as expected as I am deploying the powerful transformer based architecture. (compared with classical ML and RNN frameworks)

1.2 Results and Discussion:



Rank				Paraphrase Identification (PI)			Semantic Similarity (SS)			
PI	SS	Team	Run	F1	Precision	Recall	Pearson	maxF1	mPrec	mRecall
		Human Upperbound		**0.823**	**0.752**	**0.908**	**0.735**	——	——	——
1		ASOBEK	01_svckernel	0.674^{1}	0.680	0.669	0.475^{18}	0.616	0.732	0.531
	8	ASOBEK	02_linearsvm	0.672^{2}	0.682	0.663	0.504^{14}	0.663	0.723	0.611
2	1	MITRE	01_ikr	0.667^{3}	0.569	0.806	0.619^{1}	0.716	0.750	0.686
3		ECNU	02_nnfeats	0.662^{4}	0.767	0.583	——	——	——	——
4		FBK-HLT	01_voted	0.659^{5}	0.685	0.634	0.462^{19}	0.607	0.551	0.674
5		TKLBLIIR	02_gs0105	0.659^{5}	0.645	0.674	——	——	——	——
		MITRE	02_bieber	0.652^{7}	0.559	0.783	0.612^{2}	0.724	0.753	0.697
6		HLTC-HKUST	02_run2	0.652^{7}	0.574	0.754	0.545^{6}	0.669	0.738	0.611

Table 1: Top results shown in the original paper

Name	F1	Precision	Recall	Pearson	MaxF1	mPrecision	mRecall
miniLM	0.681	0.867	0.560	0.656	0.738	0.707	0.771
roBERTa	0.679	0.764	0.611	0.645	0.733	0.753	0.714

Table 2: My results generated by *pit2015_eval_single.py*

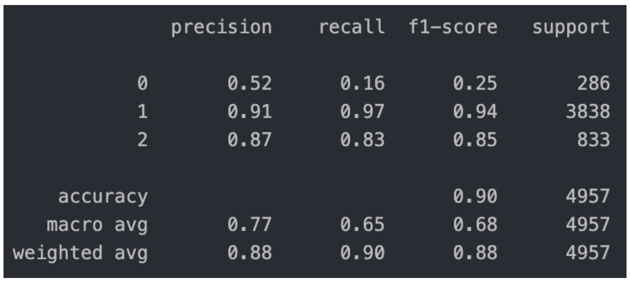
In regard to the metrics for degree outputs (semantic similarity), both models performed better than all models shown in the paper. This is as expected, as this approach is trained toward minimizing the difference between our similarity prediction against the golden similarity. Moreover, I did not fine-tune these models. I hypothesize that with fine-tuning, the results will be significantly better. With the binary outputs (paraphrase identification), my results were a bit mixed. I hypothesize that due to short training time and lack of fine-tuning, the models were not completely confident in its prediction (resulting in a lot of “middle predictions (0.4-0.6). Consequently, there exist false positives and false negatives. To combat this issue, if I were to continue working on this task, I would reduce the number of “debatable” answers to balance out the dataset. Overall, this approach took ~ 3 hours to generate the final results.

Task 2: Offensive Hate Speech Classification

Initial Thoughts: The main goal in this task is to filter out offensive, discriminatory, abusive, etc. languages. This list of “banned” words/languages will be constantly changing over time. Moreover, I would like to minimize the number of false negatives and positives. More importantly, I have a limited time and computational resources as stated. To summarize, this task boils down to an offensive/discriminatory classification task. We are allowed to use any dataset; thus my first task was to look for a viable dataset. I wanted to look for a dataset that consist of different groups of negative languages (racist, sexist, homophobic, nationalism, etc.) where I could filter the dataset based in this group. The main idea is to have this dataset to deal with the “constantly changing over time” aspect. With different groups of potential “black list”, I could calculate the average semantic differences between these groups. Consequently, if new policies for black list comes out, I will have a “ceiling” of distance. In other words, even if my model does not account for this new policy, I can still have a threshold of distance to potentially filter out according to the new policy. Additionally, I also thought about caching for these semantic textual differences to combat the increasing traffic volume.

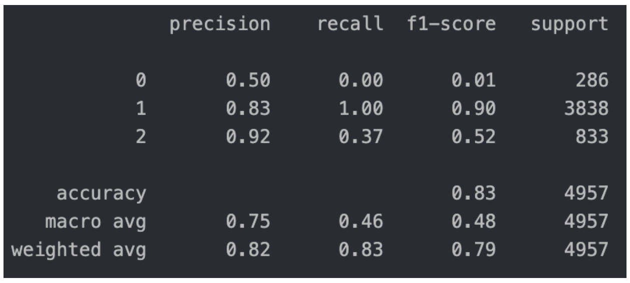
However, with the limited time that I have, I couldn't find a suitable dataset. Consequently, I decided to use the most basic/popular dataset: Hate_speech_offensive dataset. This dataset consisted of 3 labels: 0 (hate speech), 1 (offensive), and 2 (neither). In addition to this dataset, I also examined the HateXplain dataset (to potential increase performance and explainability to the predicted decision). However, time limitation proved to be infeasible to keep experimenting with this dataset. Thus, with the motto of MVP is better than nothing, I stopped researching, and started with the hate_speech_offensive dataset to produce some results. Overall, I was to restart this task, I would not spend as much time researching dataset due to the time restriction

First Approach: With the main restriction being computational resource and time (as mentioned in the README), my first idea was to leverage classical ML methods while transforming the sentences into some embedding (in this case, I used TFIDF). After cleaning the tweets, I simply employed classical ML classification framework such as: LogReg, NaiveBayes, and XGBoost.



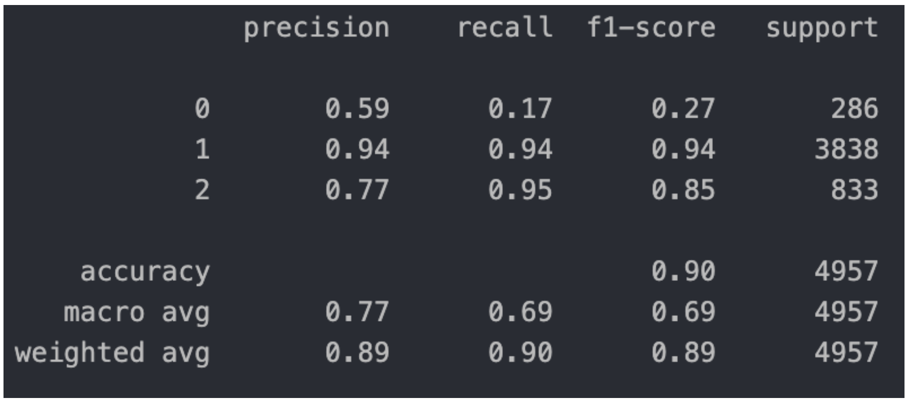
```
              precision    recall  f1-score   support

           0       0.52      0.16      0.25       286
           1       0.91      0.97      0.94      3838
           2       0.87      0.83      0.85       833

    accuracy                           0.90      4957
   macro avg       0.77      0.65      0.68      4957
weighted avg       0.88      0.90      0.88      4957
```

LogReg



```
              precision    recall  f1-score   support

           0       0.50      0.00      0.01       286
           1       0.83      1.00      0.90      3838
           2       0.92      0.37      0.52       833

    accuracy                           0.83      4957
   macro avg       0.75      0.46      0.48      4957
weighted avg       0.82      0.83      0.79      4957
```

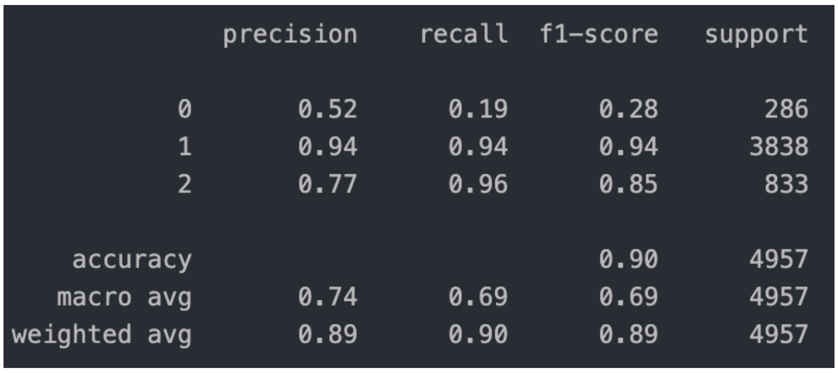
NaiveBayes



```
              precision    recall  f1-score   support

           0       0.59      0.17      0.27       286
           1       0.94      0.94      0.94      3838
           2       0.77      0.95      0.85       833

    accuracy                           0.90      4957
   macro avg       0.77      0.69      0.69      4957
weighted avg       0.89      0.90      0.89      4957
```

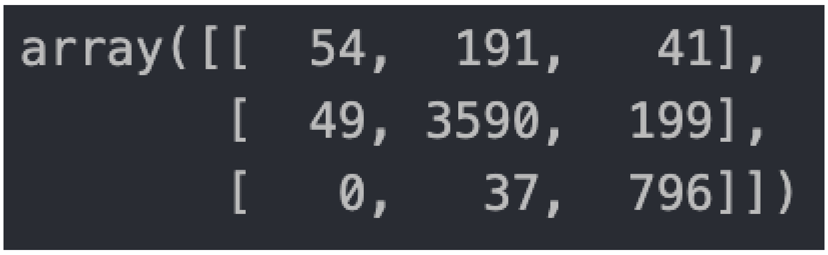
XGBoost

Picking the best performing classifier (in this case, XGBoost), I performed some grid-search to find the best hyperparameters. I chose to employ Random grid-search due to our computational resource restriction. With limited resources, normal grid-search would simply be too time-consuming/infeasible. After the random grid-search, I achieved the following results, which was only marginally better:



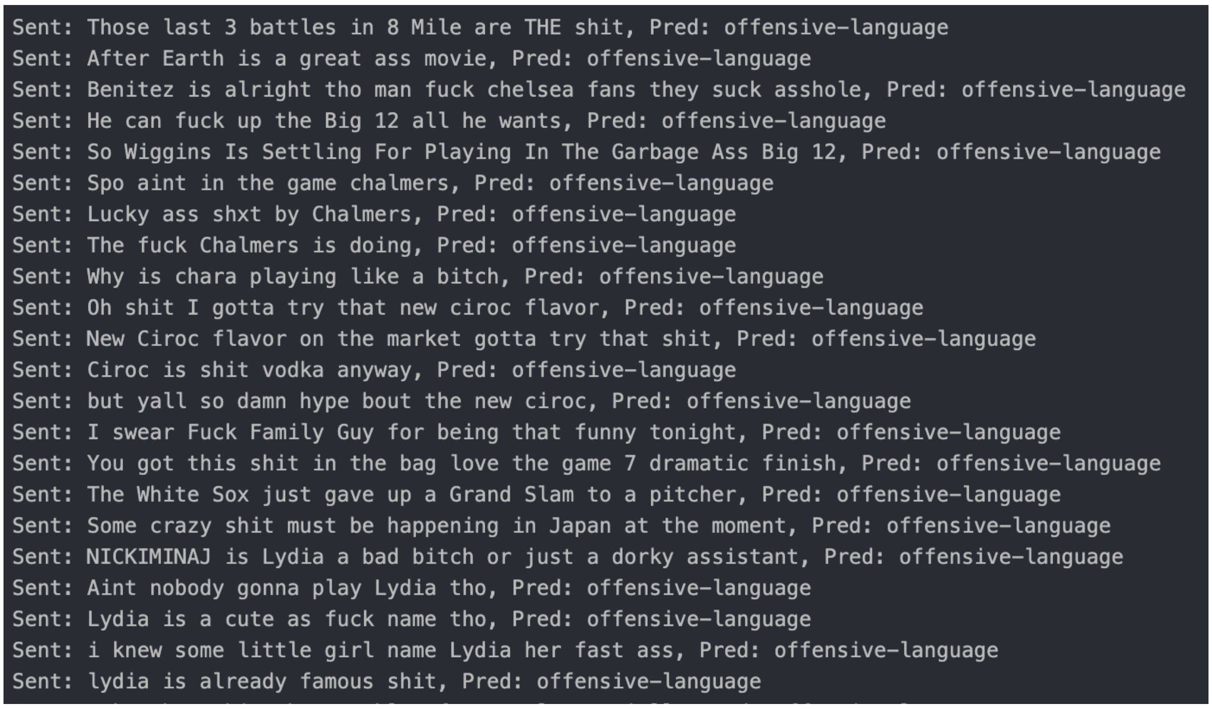
```
              precision    recall  f1-score   support

           0       0.52      0.19      0.28       286
           1       0.94      0.94      0.94      3838
           2       0.77      0.96      0.85       833

    accuracy                           0.90      4957
   macro avg       0.74      0.69      0.69      4957
weighted avg       0.89      0.90      0.89      4957
```

Due to imbalances of the dataset, the classifier hardly predicts the first class (hate speech). To further inspect this behavior, I have included the confusion matrix below:



```
array([[  54,  191,   41],
       [  49, 3590,  199],
       [   0,   37,  796]])
```

Overall, the final "fine-tuned" XGBoost's performance was okay, but nothing spectacular. I further tested out this classifier on the SemEval datasets. The following are the sentences predicted to be offensive.



```
Sent: Those last 3 battles in 8 Mile are THE shit, Pred: offensive-language
Sent: After Earth is a great ass movie, Pred: offensive-language
Sent: Benitez is alright tho man fuck chelsea fans they suck asshole, Pred: offensive-language
Sent: He can fuck up the Big 12 all he wants, Pred: offensive-language
Sent: So Wiggins Is Settling For Playing In The Garbage Ass Big 12, Pred: offensive-language
Sent: Spo aint in the game chalmers, Pred: offensive-language
Sent: Lucky ass shxt by Chalmers, Pred: offensive-language
Sent: The fuck Chalmers is doing, Pred: offensive-language
Sent: Why is chara playing like a bitch, Pred: offensive-language
Sent: Oh shit I gotta try that new ciroc flavor, Pred: offensive-language
Sent: New Ciroc flavor on the market gotta try that shit, Pred: offensive-language
Sent: Ciroc is shit vodka anyway, Pred: offensive-language
Sent: but yall so damn hype bout the new ciroc, Pred: offensive-language
Sent: I swear Fuck Family Guy for being that funny tonight, Pred: offensive-language
Sent: You got this shit in the bag love the game 7 dramatic finish, Pred: offensive-language
Sent: The White Sox just gave up a Grand Slam to a pitcher, Pred: offensive-language
Sent: Some crazy shit must be happening in Japan at the moment, Pred: offensive-language
Sent: NICKIMINAJ is Lydia a bad bitch or just a dorky assistant, Pred: offensive-language
Sent: Aint nobody gonna play Lydia tho, Pred: offensive-language
Sent: Lydia is a cute as fuck name tho, Pred: offensive-language
Sent: i knew some little girl name Lydia her fast ass, Pred: offensive-language
Sent: lydia is already famous shit, Pred: offensive-language
```

It seems like the classifier recognized any curse words as offensive without any connotations (ie Lydia is a cute as fuck name tho). Even if the sentence has a good meaning; the classifier will predict it as offensive if it includes any curse words. Thus, if there is GPU/computational resource, I propose to employ attention based architectures. With the attention mechanism, I believe this behavior will be significantly reduced.

Second Approach: With more resources and time, I propose to use some transformer based architecture to classify our tweets. I only had enough time to create and train a pipeline based on *roBERTa-base* without many changes. Thus, the performance is subpar as shown below (via *trainer.evaluate()*)

{'eval_loss': 0.2431873381137848, 'eval_f1': 0.8539439176921525, 'eval_accuracy': 0.8539439176921525, 'eval_recall': 0.8539439176921525, 'eval_runtime': 5.2193, 'eval_samples_per_second': 949.748, 'eval_steps_per_second': 118.79}