NLP Project 4

Submitted by: Sudhanshu Khoriya (sdk225), Soham Ray (sr2259), Saurabh Yadav (sky36), Dilip Reddy (dr589)

Kaggle:

Team Name : **Team Name (**ranked 8th in both Part A and Part B)

Part A:

In this section we will describe the classification of correct ending for ROC Stories with custom defined features which will be elaborated in the subsequent sections.

Description of our System:

We have used Logistic Regression for Classification. To do this effectively, we have performed rigorous data preprocessing and feature engineering, as explained in section 1 and 2. Next, we experimented with multiple classifiers and obtained best results (71.7% test accuracy) with Logistic Regression.

1) **Pre-processing:**

i) Removing punctuation and stop words:

After reading the data from the csv file, we created an array with the 4-ss(concatenating them together) , ending1, ending 2 and labels separately for the training, validation and test data. We then removed the punctuation (apostrophes, commas, full stops etc) and the stop words from them to make them more useful for use along with word embeddings to give a better context and inference.

ii) Tokenization:

After data was preprocessed, we tokenized the sentences by splitting them with white spaces to get all the words in the sentences and stored them in a list for each 4-ss and the endings

iii) Lemmatization:

The words were lemmetized in order to remove the inflecting forms and consider them as the same. This was necessary in order to get the correct inference of the words and avoid duplication of the same words based on different inflected form.

2) **Features Engineering:**

i)Word embeddings:

We represented all the words in the corpus using Glove word embeddings (100d). We selected these features since we need the inference or meaning of a sequence of sentences. One hot encoding can be a valid feature for representing all the words in the vocabulary of the document but word embeddings give us more meaningful feature and they can be used for finding similarity between different words considering all their synonyms and parts of speech.
We averaged the word embeddings for each word in 4-ss and the endings by the number of words in the corresponding sentences and represented them as 100-d vectors individually.

ii) Sentiment:

The next feature we used was the sentiment of the 4-ss and the endings individually. The sentiment was extracted by using Nltk: Vader library which gives the compound sentiment in numerical form in range of -1(very negative) to 1(very positive). We appended each 4-ss story and the endings with their sentiments. This is an important feature as it gives us the quantitative sentiment of each of them and it can give us some insight on how similar the sentiment of the endings are with respect to the 4-ss.

iii) Similarity between the word embeddings vector and the ending vector:

The final feature that we implemented was the cosine similarity between the word embeddings vector of the 4-ss and each of the endings.

The final feature vector was a concatenation of the 4-ss , the sentiment of 4-ss, a delimiter token, word embedding of the ending vector, sentiment of that ending and the cosine similarity of the ending and 4-ss along with the label of the ending (0 for wrong, 1 for right).

So our training feature sample space consisted of 2994 feature vectors . Similarly development set had 748 feature vectors and test data had 1781*2 = 3742 features.

iv) N-grams:

We have used trigrams for best results. These trigrams are then converted into their word embeddings which are used to create the feature vectors.

3) **Model training and validation :**

The final model which was used for classification was logistic regression (sklearn_linearmodel). We trained the logistic regression model with 2994 training samples with the features described above and the labels for these samples were 1 for correct ending and 0 for wrong ending.

For training we used the above feature set for all the samples and used l2 loss for calculating the loss penalty and random seed 0 .

For validation: We calculated the prediction probability of the 4-ss + ending1 and 4-ss + ending2 for each sample in validation set, and then the logistic regression classifier gives us two values for each of the example (probability of label of example being zero and probability of it being 1) .

We now do the logit comparison of the four probabilities we just got for each example.
Let's consider the probability of ending1 are : $p_{10,}$ p_{11} and probability of ending2 are : p_{20}, p_{21} . Now we first consider the maximum of the two probabilities of each ending.
If p_{10} is maximum for ending 1 and p_{21} is maximum for ending2 then both the probabilities are in unison and we can say that ending2 is the correct ending as it's probability of being 1 is more. Similarly, when p_{11} is max for ending1 and p_{20} is max for ending2 then we can say that ending1 is correct ending. When p_{11} is max for ending1 and p_{21} is max for ending2 then in this case we compare them and the one with greater value gets the label for being correct ending.

This was also done for the test set and the labels were predicted accordingly.

Justification for selection of features:

The above features were selected because they give us the most meaningful inference of the sentences . The **word embeddings** can be used to get the information about the words in each sentence. Just concatenating the word embeddings for each word made it difficult to assign appropriate weights for the features because of unequal length of sentences. The averaging of word embeddings are used in most NLP tasks and are found to be effective.

The **sentiment** of 4-ss and the endings are extracted using nltk.vader library. It takes the words in the sentence in consideration and we can use it to determine the similarity of the sentiments.

The **cosine similarity** between the word embeddings vector serve as another feature to match the similarity between the words of the 4-ss and endings as another linguistic feature as we want to associate the correct ending with the original story. The choice of our features gives more importance to the words that matter the most in the sequence and as this is a sequence inference task we thought these features would make the most sense.

The other linguistic features that we used were **n-grams** as they provide collocation information. We used trigrams so maximum context is captured without overfitting the model.

Detailed Feature Representation of a single feature:

4-ss: [Rick', 'grew', 'up', 'in', 'a', 'troubled', 'household', 'He', 'never', 'found', 'good', 'support', 'in', 'family', 'and', 'turned', 'to', 'gangs', 'It', 'wasnt', 'long', 'before', 'Rick', 'got', 'shot', 'in', 'a', 'robbery', 'The', 'incident', 'caused', 'him', 'to', 'turn', 'a', 'new', 'leaf']

Feature:
```
[ 5.37384404e-02  2.88886231e-02  1.00498537e-01 -1.79890571e-01
   5.27575580e-02   2.66910378e-01   -9.88938109e-03   1.14827886e-01   -5.46743528e-02   1.47861314e-01
-9.38438459e-02   1.01051131e-01   1.91528177e-01   -6.85043623e-02   -2.15952246e-01   -3.06235212e-01
……………………………………………..02  -2.55888332e-01  -2.75413507e-01   2.02924898e-02  -1.58633593e-01
-1.00566561e-01   2.93264166e-02   -3.63769986e-01   -3.99726278e-04   -1.37419641e-01   -1.01143766e-01
3.42953373e-01  3.94011672e-02]
```

Sentiment 4-ss: [-7.69300000e-01]

Delimiter: [1.0000e+03]

Ending: ['He', 'is', 'happy', 'now']
```
[-3163651  0.1118739   0.15492464 -0.44264387 -0.17591108 -0.02900583   0.27679537 -0.34716178 -0.07044077
-0.03900992  0.34994902 -0.12490323   0.01035482 -0.5432933   0.05908585 -0.56849692  0.21187777  0.68489633
……………………………………….-0.12242037 -0.25332425 -0.16474836 -0.26990334  -0.10871844 -0.12943376
0.03957151  0.05876928 -0.41701952  0.08354924
 -0.05595466 -0.35018687  0.21445869  0.31220371]
```

Sentiment of ending : [5.71900000e-01]

Cosine similarity : [8.36998964e-01]

Linguistic features:
Features we've used are:

- N-grams
- Sentiment
- Cosine Similarity
- Word embeddings

These have been explained in detail in the Description of Model, Section 2.

Analysis

1. **Quantitative Analysis**
   - 1.1. Best Model: Logistic Regression (Test Accuracy: 71.7% | Val Accuracy: 69%):
     - 1.1.1. Features Used: [Word Embeddings, Sentiment, Cosine Similarity, N-grams]
   - 1.2. Other Models tried:
     - 1.2.1. Logistic Regression (Test Accuracy: 65.35% | Val Accuracy: 65%):
       - 1.2.1.1. Features Used: [Word Embeddings, Sentiment, Cosine Similarity]
     - 1.2.2. Logistic Regression (Test Accuracy: 56.35% | Val Accuracy: 57%):
       - 1.2.2.1. Features Used: [Word Embeddings - individual (not averaged), Sentiment, Cosine Similarity]
     - 1.2.3. Logistic Regression (Test Accuracy: 66.35% | Val Accuracy: 68%):
       - 1.2.3.1. Features Used: [Word Embeddings, Sentiment, n-grams]
     - 1.2.4. Random Forest (Test Accuracy: 64.7% | Val Accuracy: 65%):
       - 1.2.4.1. Features Used: [Word Embeddings, Sentiment, Cosine Similarity, N-grams]
     - 1.2.5. Multi-Layer Perceptron using Scikit (Test Accuracy: 62.5% | Val Accuracy: 66%):
       - 1.2.5.1. Features Used: [Word Embeddings, Sentiment, Cosine Similarity, N-grams]
     - 1.2.6. FFNN using Pytorch (Test Accuracy: 66.9% | Val Accuracy: 70%)
       - 1.2.6.1. Features Used: [Word Embeddings, Sentiment, Cosine Similarity, N-grams]

   From our quantitative analysis we can see that logistic regression gave us the best results when used with averaged word embeddings, sentiment, cosine similarity, and n-grams.

   We can also see what features are important by doing an ablation study. Without n-grams, accuracy drops. However, dropping cosine similarity doesn't make much of a difference, even though adding it increases accuracy a little. We also observe that not using averaged word embeddings significantly drops accuracy.

   Across models, we see that Logistic Regression and Random Forest perform the best as compared to feed forward Neural Networks.

2. **Qualitative Analysis**

The weight vector :
[0.007960283162239094, 3, 0.05279745295853479, 0.09873419100391506, -0.09415412863316933, -0.07652206439531735,0.016638079096461932,…….0.017624280585021867, 0.017624280585021867, **0.16804304383209675(s1)**, **-0.000004304383209675(de)**, 0.0016804304383209675, 0.0196910850545087,

0.0196910850545087, 0.0196910850545087, ……………………………………...0.017624280585021867, 0.017624280585021867, 0.017624280585021867, 0.011868445951142283, 0.011868445951142283,, -0.01734692822000819, -0.018632453440255107, 0.01494681327436362, **0.081494681327436362(s2), 0.003685640165891488 (cs)**]

From the distribution of weights as described above, the highlighted weights are: s1,s2 - sentiment of the 4-ss and ending respectively, de- weight of delimiter, cs - cosine similarity
i) From the above distribution of weights we found out that top 50 weights consisted of features from the word embeddings of 4-ss and word embeddings of ending and the sentiment of 4-ss and sentiment of ending.
ii) Cosine similarity of 4-ss and ending vector does not constitute the top 50 weights and has less impact on the decision of the classifier.
iii) The weight of the delimiter was quite negative. The feature of delimiter has no impact. It has a negative impact as per the weights and is considered useless in this case.

Our model predicts the following sample correctly:

Tim wanted to travel overseas.	He had never been on an airplane before.	He decided to travel to China by air.	He actually enjoyed the flight!	Tim vowed never to get on another plane again.	Tim started traveling a lot of places by airplane.

Looking at the features for this example, the word travel occurs in multiple places in the 4ss and also occurs in the correct ending. Thereby, n-grams, cosine similarity and word embeddings are helping. Additionally, the incorrect ending has a negative sentiment as compared to the 4ss (positive) and the correct ending (neutral).

The incorrect prediction according to our model is:

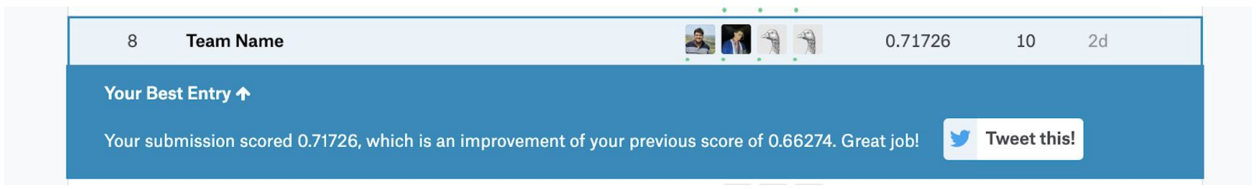
Fred liked mystery novels.	He used to read at least one every month.	When his friends found out, they all bought him one for his birthday.	But unfortunately, he started getting irritated by the attention.	Fred told his friends he was taking a break from mystery novels.	Fred started a mystery novel book club for his friends.

The features for the above example 4-ss has the n-grams features and the words that are most used are mystery novel and book. The sentiment of the example is neutral and it matches sentiment of both the endings. The second ending has more words in common , so it's word embeddings and cosine similarity matches closely with the 4-sentence story so it is classified as the correct ending by our classifier. The correct ending should have been ending1.

.
Further improvement for the model:
For further improvement, we could try RNN based approaches that store sequential information. Specifically, seq2seq models would fit the best. Another direction could be using LSTM's in a siamese network.

These models can be used to generate the next sentence. Then, we can measure the distance (using manhattan distance or cosine similarity) between the generated sentence and the endings to find the more fitting ending.

Kaggle:


8 Team Name 0.71726 10 2d

Your Best Entry

Your submission scored 0.71726, which is an improvement of your previous score of 0.66274. Great job!

Libraries Used: [Numpy, NLTK, Scikit, Pytorch, Pandas, Spacy]

Part B:

The script for the project is inspired by a PyTorch implementation of OpenAI's finetuned transformer language model[1]. Here the authors of the project have made the implementation of the TensorFlow code that is provided by OpenAI's paper "Improving Language Understanding by Generative Pre-Training"[2] by Alec Radford. This implementation comprises a script to load in the PyTorch model, with the weights pre-trained by the authors.

We have used the pre-trained version of OpenAI-GPT-2 weights by cloning Alec Radford's repo and placing the model folder in the implementation folder. The model creates the hidden states of the transformers.

Then the model is used as the transformer language model with OpenAI's pre-trained weights. We had to make a few changes to a couple of files in the pre-trained model, such as in train.py, datasets.py, analysis.py, and utils.py.

Some key changes:

- Train.py →
    - We modified the "predict function". The predict function would now take values from dev dataset as well and we also modified the way data is being written in the final csv file.
    - We also fine-tuned the default values of the hyperparameters focussed on the current dataset. We ran multiple sessions of the program and selected values for no_of_epocs, learning_rate, hidden layers that performed better in dev set and test data set without overfitting the models. These values for hyperparameters can also be passed from the command line itself. Thereby making it flexible.
    - We modified the "log" function so that we can fetch the dev set accuracy and cost-plus training set accuracy and cost for each run and each epoch as well. (loss for one run and cost for one epoch)
    - Though few have trained the model on CPU and not GPU, without having left the option open and flexible.

- Datasets.py →
    - Added a new function "_rocstories_test" for pre-processing the test dataset. The model was first using train_test_split of sklearn.model_selection to split the dataset into training and validation data. Modified the function to take the dev set separately along with the training set.
    - Modified the preprocessing of the dev set, training set, and the test set.

- Analysis.py →
    - Modified the function "rocstories" to calculate the final accuracy values for the dev set.

Learning Curve for the Best System

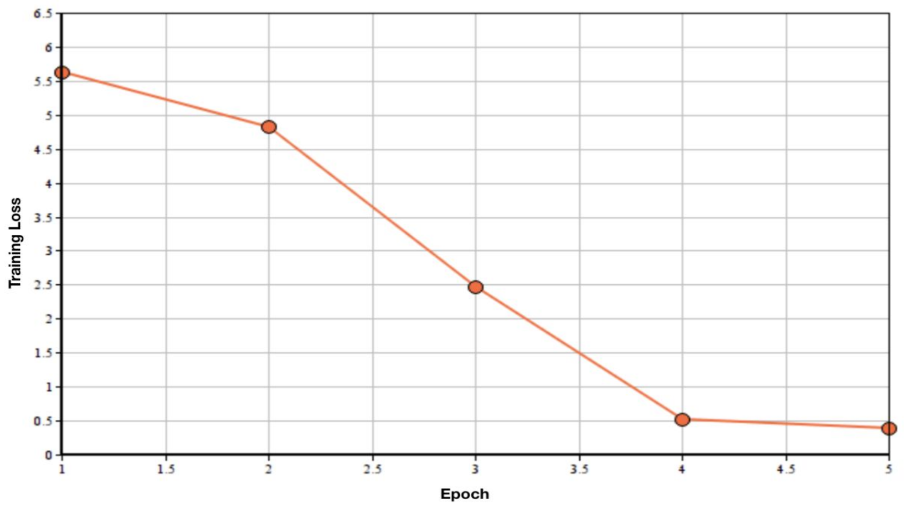


Figure 1: Training Loss vs. Epoch

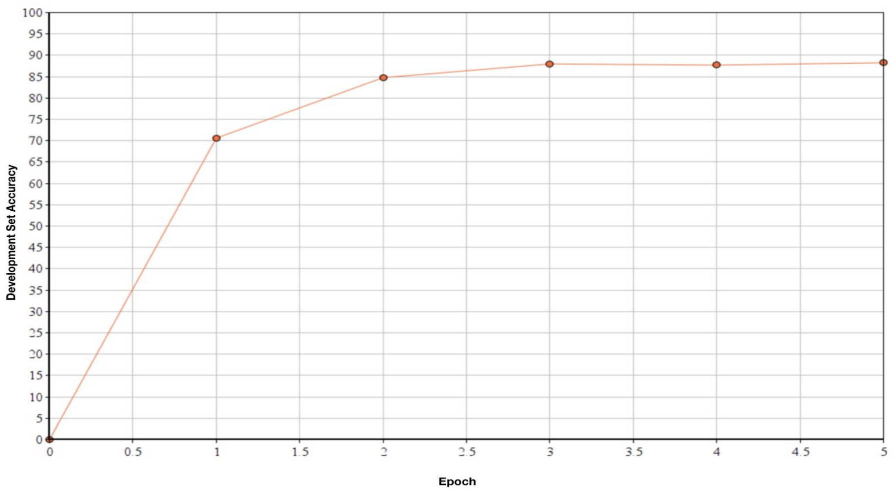


Figure 2: Development Set Accuracy vs. Epoch

Quantitative and Qualitative Analysis of the Development Set

Quantitative Analysis
Let us consider two epochs (3 and 5) and showcase how changing the learning rate and the number of layers affects the accuracy.

In epoch-3, a learning rate of 6.25e-05 and a layer size of 12 yields an accuracy of 85.16. Let this be a benchmark.

In epoch 5, a learning rate of 6.25e-05 and a layer size of 12 yields an accuracy of 86.17.
However, in comparison, with a lower learning rate of 2.0e-05 and a layer size of 24, we get a higher accuracy of 88.24.

It can be inferred from the above observations that:
(i) Number of layers implemented is directly proportional to a higher accuracy of prediction.
(ii) Learning Rate is inversely proportional to the accuracy of prediction.

However, on average, doubling the number of layers, from 12 to 24, did result in longer execution times of upto 50-65 minutes more than the time required for execution with layer size 12 (for 5 epochs, in this case).

Qualitative Analysis
OpenAIGPT-2 [Double-headed transformer] performs better than Logistic Regression since it conditions from both directions (left and right) to get the context in both directions on all layers. On the other hand, Logistic Regression is void of any context. Logistic Regression also outputs a binary value, whereas OpenAI outputs a probability value, which is more fine-grained and gives a higher accuracy value.

Example: *(Line 309 from dev.csv*)
"Teresa was an artist. She made a beautiful sculpture of an angel. She entered it into an art contest. The judges liked the sculpture very much."
Fifth Sentence Option-1: She won the contest and was proud of herself.
Fifth Sentence Option-2: She was disqualified from the contest.

The correct fifth sentence is Option-1. OpenAI's GPT-2 is able to predict this accurately due to the sequential context. For example, the judges liking "the" sculpture could refer to any sculpture, in general, but is made clear only from the contextual information that the sculpture in question is Teresa's. To emphasize the positive outcome, i.e., Teresa winning the contest, the fact that Teresa was an artist reinforces the possibility of her winning the competition. The word "beautiful" from sentence-2 also adds to the context and helps it in determining the outcome of the contest.

Quantitative and Qualitative Analysis of Part A vs Part B

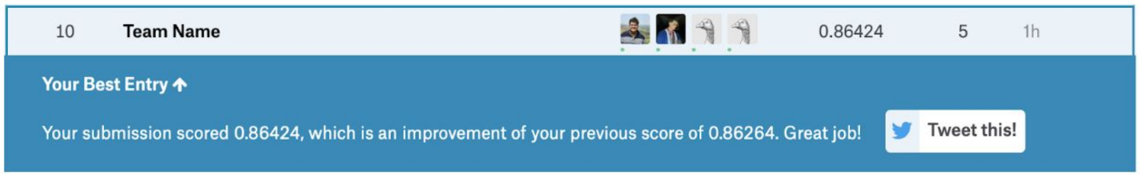
Quantitative Analysis
For part A, the test accuracy we achieved using Feed forward neural networks, and multiple linear classifiers was **71% at best**. However, for OpenAI-GPT-2, we get test accuracies from **82% to 86%**. This is as seq2seq RNN's take sequential information into context that other classifiers are unable to. This greatly improves accuracy. Additionally, OpenAI-GPT-2 uses neural attention which pays heed to important parts of the 4ss, further improving accuracy.

Qualitative Analysis
Here, we will consider one example from our development set where Logistic regression fails to classify it correctly and OpenAI-GPT-2 classifies the correct ending for it.

1a9a1b68-b971-4eec-9706-4d109934e8be	Fred liked mystery novels.	He used to read at least one every month.	When his friends found out, they all bought him one for his birthday.	But unfortunately, he started getting irritated by the attention.	Fred told his friends he was taking a break from mystery novels.	Fred started a mystery novel book club for his friends.	1

This example becomes difficult for our feature based model i.e. Logistic Regression to classify properly as it relies on the occurrences of common words, n grams , sentiment and cosine similarity. As these features are better for ending2 in case of Logistic Regression , so it classifies it incorrectly but in case of OpenAI-GPT , it considers the sequence of the sentences, so it clearly infers the meaning of the sequence of both endings and then decides on the better fit.

Kaggle Submission
[Team name is "Team Name"]


10 Team Name 0.86424 5 1h

Your Best Entry ↑

Your submission scored 0.86424, which is an improvement of your previous score of 0.86264. Great job!

Figure 3: Part-B Kaggle Submission

References

[1] Grégory Châtel, Thomas Wolf, Tom B. Brown, Sylvian Gugger and Sosuke Kobayashi. A PyTorch implementation of OpenAI's finetuned transformer language model with a script to import the weights pre-trained by OpenAI. In *https://github.com/huggingface/pytorch-openai-transformer-lm*.

[2] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. In *https://openai.com/blog/language-unsupervised/*.

Student Contributions:
Soham: Part A modelling and Feature Engineering, Experimentation, Report
Sudhanshu: Part A modelling and Feature Engineering, Experimentation, Report
Dilip: Part B modelling and Fine-Tuning, Experimentation, Report
Saurabh: Part B modelling and Fine-Tuning, Experimentation, Report

Feedback:
Project was interesting and helped understand classroom concepts. Part A also gave us an opportunity to think for ourselves and use everything we've learnt so far.