CS 5740 Project 4 report

Kaggle team name: code monkas

Team members: Pu Zhao (pz253), Shunqi Mao (sm2784), Suet Cho (sc2299)

Introduction and Project Goal

Understanding complete stories is a challenging task for NLP since the it demands a good comprehension on story sentiments, word representations, coherence and so on. In this project, we will work with ROCStories and to choose the correct ending to a four-sentence story given two options for the ending using supervised machine learning models. In the first part of the project, we applied our NLP knowledge to build a feature-based classifier for ROCStories. We recycled, analyzed and improved different machine learning models that we implemented from the previous projects while manually engineered different feature on the dataset. In this part, we are looking to explore proper feature spaces while optimizing the performance of the supervised learning model. In the second part of the project, we used deep contextualized BERT model to classify the correct story ending.

Part A: Feature-based classifiers

First Model: Naïve Bayes Classification Model

In our first modelling attempt, we came up with bag-of-words classification models since this is the most intuitive and flexible model in terms of text classification. A bag-of-words is a representation of text that describes the occurrence of words within a document which consists of a vocabulary list of known words and a measure of the presence of known words.

We are hoping to capture the pattern of a correct story and so we preprocess the data in a different way. From the dataset, we are given the first four sentences, a correct ending sentence and an incorrect ending sentence given the labels. Therefore, we duplicate the training data into two set; in the first set, we append the correct story ending to the first four sentences and assign it with a positive label, in the second set, we append the wrong story ending to the first four sentences and assign it with a negative label. After processing the data, we now have a training set consist of the words of the story as the features and the validity of the story as the label.

We reused the knowledge classification model that we implemented in project 1 where we train feature-based Naïve Bayes classifier for text classification. The model builds off the assumption that the features are independent from each other. As we observed previously, even though bag-of-words when the best on its own, the Naïve Bayes classifier performs the best when we also incorporate other features such as bag-of-bigrams, bag-of-trigrams, length of the text and so on. We assumed that incorporating more higher degree n-grams as features as additional linguistic features can capture more information on the training data. Therefore, we trained a Naïve Bayes classifier using bag-of-words, bag-of-bigrams, bag-of-trigrams, length of the story as the feature spaces with the validity of the stories as the labels.

As we perform cross validation using the validation set, we only obtain validation accuracies between 50~52%. The result is not satisfactory as in some cases it performs the same as blind guessing. Quantitatively, we hypothesized that having roughly 1500~ training data is not sufficient to train the classification model. On the other hand, the nature of the stories could be really different and therefore there will be a lot of first seen words and unknown words which will not add much value in terms of training the model. Qualitatively, we realized that using n-gram as modeling feature implies using Markov's assumption. In our case, for each word, the window is only limited to the preceding and succeeding three words; in another word, the first three sentences of the stories would be trivial in terms of the last sentence prediction. With these observations, we have concluded that we should use a supervised learning model that is able to capture the information from the entire story and does not make independence assumptions on the features.

Second Model: Recurrent Neural Network

Description of the System

Since we want to capture the information from the entire story, we chose to use recurrent neural network as out model. It captures the past and its prediction are influenced by what it learned from the past. On the other hand, recurrent neural networks are also designed to take a series of input with no predetermined limit on size.

In terms of the representation of the data, since we want to capture the relational information in the sequence, we also repeated the training data in a way such that we append the correct story ending to the first four sentences and assign it with a positive label of 1, in the next set, we append the wrong story ending to the first four sentences and assign it with a negative label of 0. After that, we converted all the words in the training set into the pretrained word embedding using Word2vec, a group of related models that are used to produce word embedding where these embedding were trained on the reconstruct linguistic context of words. Our first linguistic feature is the word embedding. As such, the initial input dimension on the recurrent neural network is the length of the individual story by the size of the word embedding vector of 100. Our second linguistic feature is the bigram perplexity of the stories. We first calculate the probabilities for each bigram in the entire corpus and then calculate the perplexity of each stories. Perplexity measures how well a probabilistic model predicts a story using the exponentiation of the entropy. The more information the bigram model gives us about sequences of the words in the stories, the lower the perplexity is. Hence, we propose that the correct stories will have a lower perplexity than the incorrect stories and thus we believe that it would be an adequate linguistic feature to distinguish between the correct and the wrong last sentence of the stories. The third linguistic features that we integrate is the length of the stories or the count of words. Since the perplexity and the length of the stories do not match the dimension of the word vectors, we match their dimensions by transforming them with padding. Therefore, the inputs of the recurrent neural networks are the number of words of an individual story, its bigram perplexity and its story length, by a dimension of 100. The outputs of the recurrent neural network would be a positive label of 1 and a negative label of 0 with a SoftMax function that produce the prediction probability.

Implementation and Reasons to Choose the Feature Set

Our first feature is the word embedding for each word in the stories. We implemented it by converting all the words in the training set into the pretrained word embedding using the Word2vec library. The reason why we used the word embedding as feature is because word embeddings allows words with similar meaning to have a similar representation in a context. Each word is mapped to one vector and the vector values are learned to resembles a neural network. Therefore, they are the essential inputs for deep learning models which are required to training our recurrent neural network. Our second feature is the bigram perplexity of each stories. We implemented functions that calculate the probabilities of the bigram, incorporate additive smoothing and perplexity. The reason why we incorporate the perplexity as a feature is because it can potentially be a feature that distinguish between the correct story ending and the wrong story ending. However, this feature is focused on and only limited toward the last sentence of the stories since the by Markov assumption, we cannot capture the meaning from words that appeared at the very beginning. Our third feature is the length of the stories. We incorporated this feature simply by calculating the length of each story and transform it to the same dimension as the word embedding with padding. As stated in the studies on linguistics, the sentences with a more positive sentiment, less negation and a longer length are all indicators of a truthful sentence. Therefore, by incorporating the length of the story as one of the features, we are hoping to capture information about the effects of the length of the story to its validity. Similarly, this feature of story length is also only limited to the last sentence of the story since the first four sentences are identical whether the ending is correct or not.

Detailed Illustration of Extracted Features

The feature space for one training example is a list consist of the word embedding for each word in the story, each with a vector dimension of 100, the bigram perplexity of the last sentence with a vector dimension of 100 after padding, and the length of the story ending sentence with a vector dimension of 100. For instances, if a story consists of 50 words, the input dimension would be 52 by 100. In **Appendix A**, we incorporated a detailed illustration of extracted features for the first story with a wrong ending. Due to the lengthiness of the feature representation of the entire training example, we illustrate the last two words' embedding with the perplexity feature and the length of the story feature.

Quantitative and Qualitative analysis

Using the development set, we can compute the validation accuracy of our model when we are training it. The recurrent neural network with three linguistic features that we developed can achieve 0.545326 validation accuracy after learning with the training samples. Qualitatively, as we proposed above, the word embedding is essential in terms of training a neural network. Along with the bigram perplexity feature and the sentence length feature, we are expecting that the model can capture the information on top of word embedding. Therefore, in the baseline model, we train the recurrent neural networks using only the word embeddings in the feature space. We assumed that the baseline model is adequate to predict the story ending as we assumed that the architecture of recurrent neural network with word embeddings alone can capture enough information of the previous words to predict the story ending. As such, when we train our baseline model, we attained an accuracy of 0.5190217. This is consistent to our hypothesis where word embedding is a significant feature in training our system. After that, we built up our model by incorporating the bigram perplexity as an additional feature. As discussed above, it can potentially be a feature that distinguish between the correct story ending and the wrong story ending because bigram captures the relationships between words next to each other. After training the model with word embedding with bigram perplexity, we attained a validation accuracy of 0.5203877. It shows that bigram perplexity is a less significant feature. Qualitatively, we assumed that the word relations in the stories were already captured by the hidden matrix in the recurrent neural network and thus adding the bigram perplexity would not add additional using information to the model. Overall, our recurrent neural network can achieve a prediction accuracy of roughly 55%.

Third Model: Application of Word Embeddings

Description of the System

Our third model is a word vector comparison model. The idea behind this is to use the information from the word embeddings to capture the meaning of the story. First, we take the average of the word embeddings in the first four sentences. Then, we compare the difference between the vector of the first sentence option and the second sentence option. After computing the differences between the word vectors, we can predict the ending sentence of the story by choosing the vector that is closer.

Implementation and Features

To implement this model, we tokenized the entire training data and then converted the words into word embedding using Word2Vec. For story, we first sum up all the word vectors in the first four sentences and then divide by the average number of the words in the story. We repeat the same procedure for the first sentence option and the second sentence option. Then, we take the difference between the story with the first sentence and the story with the second sentence to see which one is less different. The feature set for this model is simply the word embeddings that are trained on the story training data. To quantify the differences, we came up with three different features: dot product between the two vectors, the Euclidean distance between the two vectors and the cosine similarity between the two vectors.

Quantitative and Qualitative Analysis

Using the training and development set, we can compute the validation accuracy of our model when we are calculating the word vectors. In our first attempt, we predicted that the story sentence pair that has a lower dot product is the correct sentence. The reason behind this is that by calculating the dot product, we calculate the Euclidean magnitude of the two vectors which also include the cosine of the angle between them. Therefore, we assume that the story sentence pair with a smaller magnitude has a higher similarity. With this approach, we attained an accuracy of 0.52640374 which is not ideal. With our second feature, we predict the story sentence pair that has a lower Euclidean distance. The reason is that the sentence that has a shorter distance are more similar and therefore is more likely to be related to the story. After incorporating this feature, the accuracy that we got from the development set is 0.5938543. In the last approach, we predict the sentence that has a higher cosine similarity to be the correct ending sentence. Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. It measures the cosine of the angle between the two vectors. The cosine similarity is advantageous because if the two similar documents are far apart by Euclidean distance due to their difference in size, they may still be oriented closer together. In our case, this metrics is most suitable since we are comparing the vector of the story to the vector of the ending sentence. With this approach, we attained an accuracy of 0.6314698. This has verified with our hypothesis that cosine similarity is the most suitable metrics to measure the word vectors. In summary, we have concluded that our word embedding model with cosine similarity measuring metric is our best model is Part A.

Part B: Fine-tune a Pre-trained Language Model

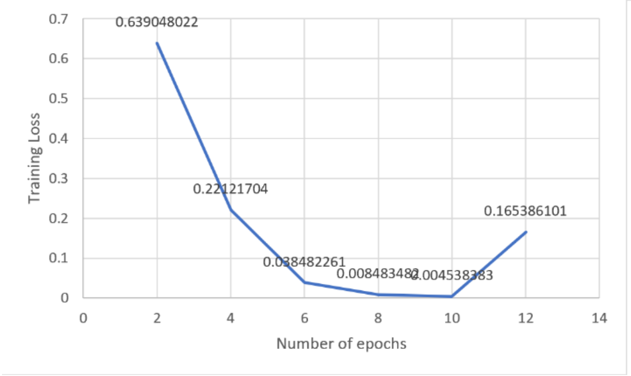
Implementation

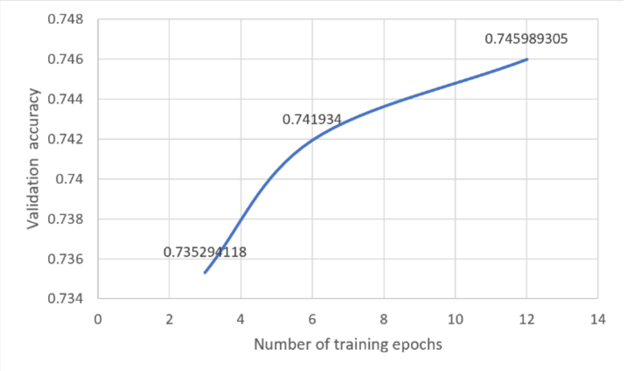
First, we set up the environment as instructed in huggingface/transformer, Github. In order to run our train.csv and dev.csv. We made modifications in utils_multiple_choice.py as well as run_multiple_choice.py.

For the 'utils' program, we made modifications mainly in SwagProcessor class. We made the return result as return ["1", "2"] in get_labels function. We modify arguments in contexts variable of create_examples function as contexts = [line[1]+' '+line[2]+' '+line[3]+' '+line[4], line[1]+' '+line[2]+' '+line[3]+' '+line[4]], ; we modify arguments in endings variable of that as endings = [line[5], line[6]], ; we modify arguments in label variable of that as label=line[7]. Replace the utils_multiple_choice.py of the original one with the new one. Replace the train.csv with a new one. Replace the val.csv with dev.csv, and rename the file name as val.csv. Rename the last column of the dev.csv as 'label'. Run as instructed in the terminal, so that we can get the development accuracy as well as the model trained by the training set.

For the 'run' program, we create a csv, and use csv writer to write the results in preds variable by iterate through elements of it. This is done after the line 'preds = np.argmax(preds, axis=1)' We overwrite data in val.csv with test.csv, and put a dummy column of label filled with arbitrary numbers after the last column. In order to get test result, we replace run_multiple_choice.py with a new one. Run as instructed, data in the output file is the prediction result.

Learning Curve Plot





Quantitative and Qualitative Analysis

Our system in Part A considers only the word embedding of words and doesn't make use of training set data to help predict test set story endings. Because calculating average word embedding of a sentence is a reasonable and valid linguistic feature, our system in Part A achieves an accuracy of 62% on the development set. However, our system in Part B uses deep contextualized BERT-Mini model to train on the training set and thus the performance on the development set is significantly higher (74%) than our system in Part A. It is also observed that increasing the number of epochs when training our system in Part B will further improve performance on the development set even when the learning rate remains unchanged. As a result, we hypothesize that BERT-Mini model is still able to learn more information from the training set if we continue tuning the learning rate or increasing the number of epochs.

We experimented with 3, 6, 12 epochs of training and learning rates of 1e-4, 5e-5, 2e-5 to see how the performance of BERT-Mini model would change along with these hyper-parameters. The performance with 5e-5 learning rate and 3 epochs is 0.73529 and the performance with 2e-5 learning rate and 6 epochs is 0.74193. Based on the similarity between these two variations, we hypothesize that a fixed learning rate doesn't help the convergence of training loss. So, we decided to use an adaptive learning rate and number of epochs to verify whether BERT-Mini can still incorporate more information from the training set. The graph of training accuracy by epochs are provided in the previous section that demonstrates the performance with an initial learning rate of 1e-4 and twelve training epochs. As expected, longer training cycles and slightly increasing learning rate boosts the performance by nearly 10%.

In addition, we experimented with batch sizes of 16 and 32 to see the change of training speed and training loss. With our best system in Part B, increasing the batch size from 16 to 32 significantly decreases training speed and doesn't have a visible impact on training loss. This is possibly because different stories in ROCStories dataset are highly irrelevant to each other and grouping examples merely increases computational complexity.

In conclusion, BERT model achieves better performance if more epochs are allowed and the learning rate is adaptive (such as decaying over a certain number of epochs) while adjusting batch sizes doesn't have an impactful effect on its performance.

Additional Information and Appendix

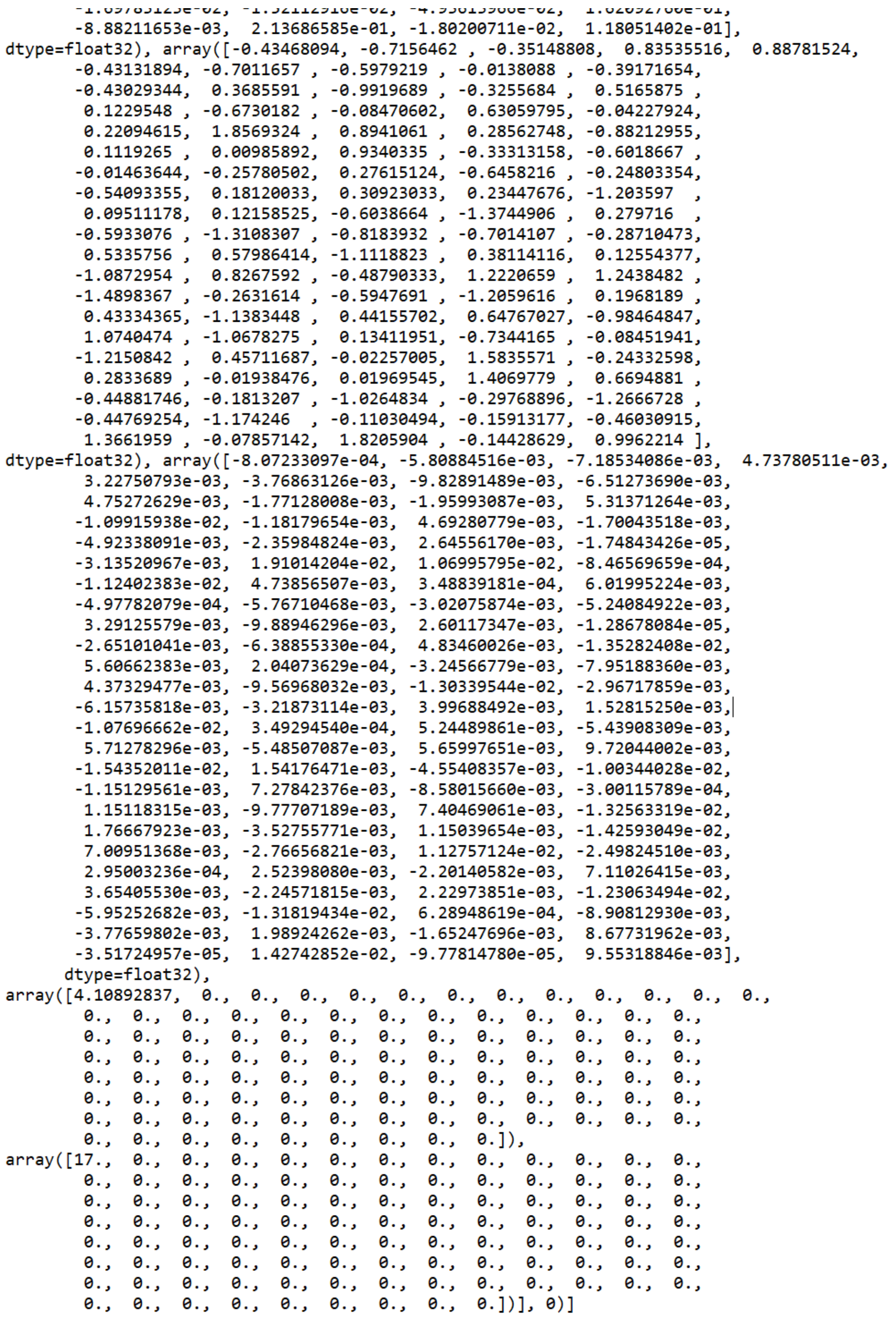
Workflow and team contributions

We implemented the different part of the projects jointly and sought for help from each other mostly on debugging. We discussed and came up with different machine learning and NLP approach for part A. After having the models in mind, we each implement some parts to extract the linguistic features. Shunqi and Suet worked on reengineering and implementing the Naive Bayes classifier, recurrent neural network, and word embedding models for part A. Pu and Shunqi put an emphasis on implementing and analyzing the BERT model from part B.

Feedback

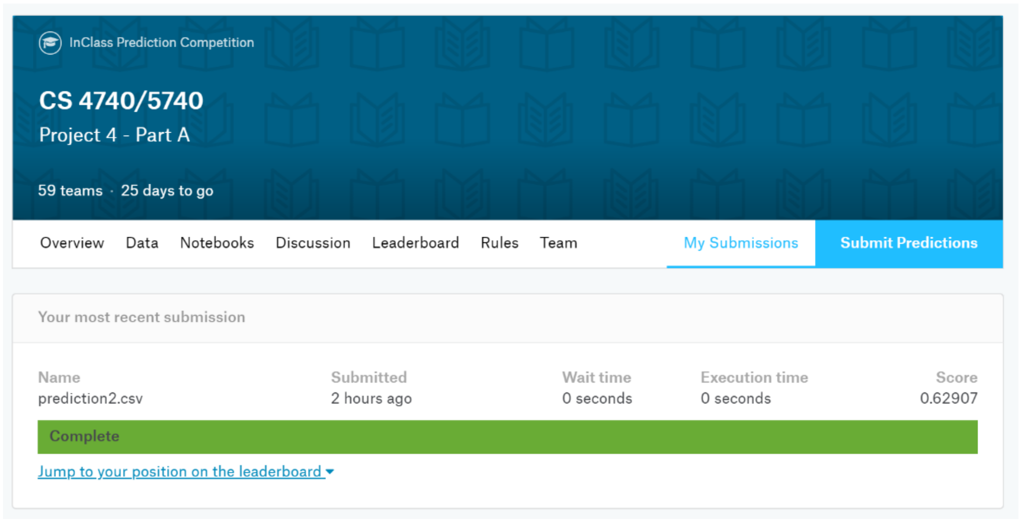
The project is interesting but also somewhat difficult because this is very open ended which challenge our understanding of the data and NLP. For part a, we built multiple models while we try to implement and train the BERT model for part b. Our team worked for approximately met up in a span of 2 weeks and worked for a total of about 48 hours on the project. This project helps with our understanding of the content as we explore different model.

Appendix A: RNN sample features



```
       -1.69783123e-02, -1.32112916e-02, -4.93613966e-02,  1.62092786e-01,
       -8.88211653e-03,  2.13686585e-01, -1.80200711e-02,  1.18051402e-01],
dtype=float32), array([-0.43468094, -0.7156462 , -0.35148808,  0.83535516,  0.88781524,
       -0.43131894, -0.7011657 , -0.5979219 , -0.0138088 , -0.39171654,
       -0.43029344,  0.3685591 , -0.9919689 , -0.3255684 ,  0.5165875 ,
        0.1229548 , -0.6730182 , -0.08470602,  0.63059795, -0.04227924,
        0.22094615,  1.8569324 ,  0.8941061 ,  0.28562748, -0.88212955,
        0.1119265 ,  0.00985892,  0.9340335 , -0.33313158, -0.6018667 ,
       -0.01463644, -0.25780502,  0.27615124, -0.6458216 , -0.24803354,
       -0.54093355,  0.18120033,  0.30923033,  0.23447676, -1.203597  ,
        0.09511178,  0.12158525, -0.6038664 , -1.3744906 ,  0.279716  ,
       -0.5933076 , -1.3108307 , -0.8183932 , -0.7014107 , -0.28710473,
        0.5335756 ,  0.57986414, -1.1118823 ,  0.38114116,  0.12554377,
       -1.0872954 ,  0.8267592 , -0.48790333,  1.2220659 ,  1.2438482 ,
       -1.4898367 , -0.2631614 , -0.5947691 , -1.2059616 ,  0.1968189 ,
        0.43334365, -1.1383448 ,  0.44155702,  0.64767027, -0.98464847,
        1.0740474 , -1.0678275 ,  0.13411951, -0.7344165 , -0.08451941,
       -1.2150842 ,  0.45711687, -0.02257005,  1.5835571 , -0.24332598,
        0.2833689 , -0.01938476,  0.01969545,  1.4069779 ,  0.6694881 ,
       -0.44881746, -0.1813207 , -1.0264834 , -0.29768896, -1.2666728 ,
       -0.44769254, -1.174246  , -0.11030494, -0.15913177, -0.46030915,
        1.3661959 , -0.07857142,  1.8205904 , -0.14428629,  0.9962214 ],
dtype=float32), array([-8.07233097e-04, -5.80884516e-03, -7.18534086e-03,  4.73780511e-03,
        3.22750793e-03, -3.76863126e-03, -9.82891489e-03, -6.51273690e-03,
        4.75272629e-03, -1.77128008e-03, -1.95993087e-03,  5.31371264e-03,
       -1.09915938e-02, -1.18179654e-03,  4.69280779e-03, -1.70043518e-03,
       -4.92338091e-03, -2.35984824e-03,  2.64556170e-03, -1.74843426e-05,
       -3.13520967e-03,  1.91014204e-02,  1.06995795e-02, -8.46569659e-04,
       -1.12402383e-02,  4.73856507e-03,  3.48839181e-04,  6.01995224e-03,
       -4.97782079e-04, -5.76710468e-03, -3.02075874e-03, -5.24084922e-03,
        3.29125579e-03, -9.88946296e-03,  2.60117347e-03, -1.28678084e-05,
       -2.65101041e-03, -6.38855330e-04,  4.83460026e-03, -1.35282408e-02,
        5.60662383e-03,  2.04073629e-04, -3.24566779e-03, -7.95188360e-03,
        4.37329477e-03, -9.56968032e-03, -1.30339544e-02, -2.96717859e-03,
       -6.15735818e-03, -3.21873114e-03,  3.99688492e-03,  1.52815250e-03,
       -1.07696662e-02,  3.49294540e-04,  5.24489861e-03, -5.43908309e-03,
        5.71278296e-03, -5.48507087e-03,  5.65997651e-03,  9.72044002e-03,
       -1.54352011e-02,  1.54176471e-03, -4.55408357e-03, -1.00344028e-02,
       -1.15129561e-03,  7.27842376e-03, -8.58015660e-03, -3.00115789e-04,
        1.15118315e-03, -9.77707189e-03,  7.40469061e-03, -1.32563319e-02,
        1.76667923e-03, -3.52755771e-03,  1.15039654e-03, -1.42593049e-02,
        7.00951368e-03, -2.76656821e-03,  1.12757124e-02, -2.49824510e-03,
        2.95003236e-04,  2.52398080e-03, -2.20140582e-03,  7.11026415e-03,
        3.65405530e-03, -2.24571815e-03,  2.22973851e-03, -1.23063494e-02,
       -5.95252682e-03, -1.31819434e-02,  6.28948619e-04, -8.90812930e-03,
       -3.77659802e-03,  1.98924262e-03, -1.65247696e-03,  8.67731962e-03,
       -3.51724957e-05,  1.42742852e-02, -9.77814780e-05,  9.55318846e-03],
      dtype=float32),
array([4.10892837,  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.,
        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.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]),
array([17.,  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.,
        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.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])], 0)]
```

Appendix B: Kaggle Submission



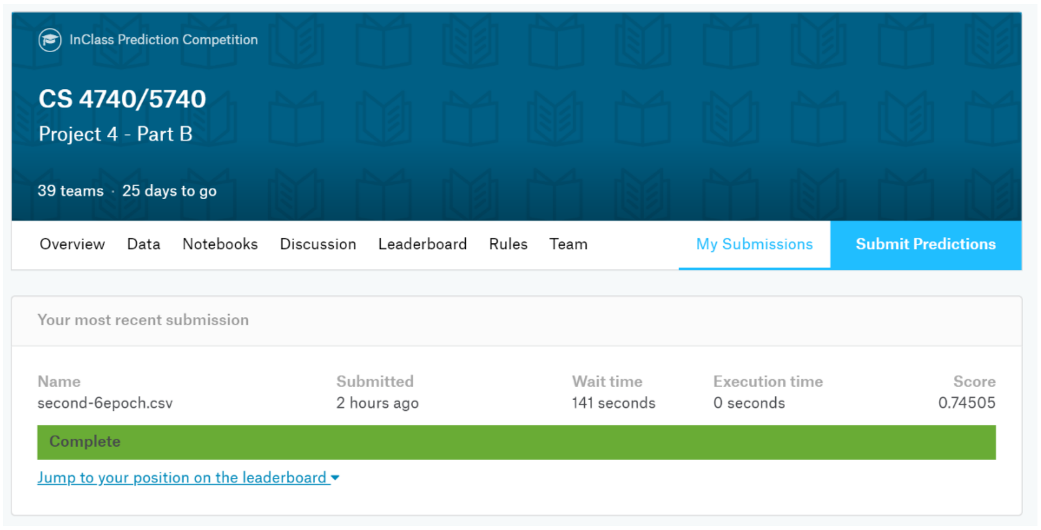
CS 4740/5740
Project 4 - Part A
59 teams · 25 days to go

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
prediction2.csv	2 hours ago	0 seconds	0 seconds	0.62907

Complete

Jump to your position on the leaderboard



CS 4740/5740
Project 4 - Part B
39 teams · 25 days to go

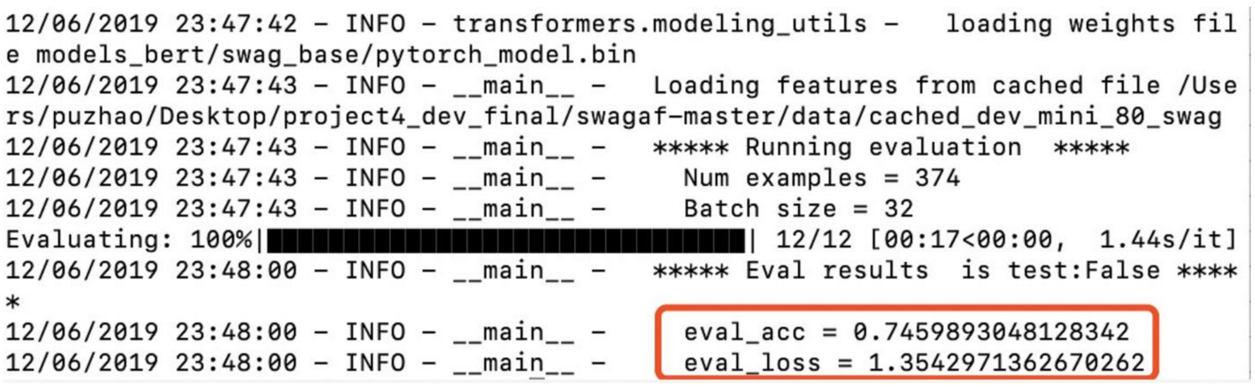
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
second-6epoch.csv	2 hours ago	141 seconds	0 seconds	0.74505

Complete

Jump to your position on the leaderboard

Appendix C: Sample BERT accuracy



```
12/06/2019 23:47:42 - INFO - transformers.modeling_utils -   loading weights fil
e models_bert/swag_base/pytorch_model.bin
12/06/2019 23:47:43 - INFO - __main__ -   Loading features from cached file /Use
rs/puzhao/Desktop/project4_dev_final/swagaf-master/data/cached_dev_mini_80_swag
12/06/2019 23:47:43 - INFO - __main__ -   ***** Running evaluation  *****
12/06/2019 23:47:43 - INFO - __main__ -     Num examples = 374
12/06/2019 23:47:43 - INFO - __main__ -     Batch size = 32
Evaluating: 100%|████████████████████████████████| 12/12 [00:17<00:00,  1.44s/it]
12/06/2019 23:48:00 - INFO - __main__ -   ***** Eval results  is test:False ****
*
12/06/2019 23:48:00 - INFO - __main__ -     eval_acc = 0.7459893048128342
12/06/2019 23:48:00 - INFO - __main__ -     eval_loss = 1.3542971362670262
```