Debates Speaker And Party Predictions

Machine Learning Project Report 097209

Technion Institute of Technology

1. Abstract:

This project addresses the problem of speaker segmentation and speaker tracking in United States presidential debate. In such a case speaker is assumed unknown and in the advanced procedure the speaker's party is the unknown.

Natural Language Processing (NLP) is a dynamic and rapidly developing field in which new trends, techniques and applications are constantly emerging. NLP extracts precise neurological, verbal and non-verbal information, and assimilates the information into useful patterns. These patterns are based on specific cues demonstrated by each individual and provide ways of determining congruency between verbal and non-verbal cues. The primary NLP modalities are characterized through word spotting (or verbal predicates cues, e.g., see, sound, feel, etc.) while the secondary modalities would be characterized through the speech transcription used by the individual. This has the practical effect of reducing the size of the search space, and greatly speeding up the process of identifying an unknown speaker.

Speaker segmentation is a part of a necessary mission in NLP. One of the most important elements of a conversation is the identity of the speaker at each sentence and the changing of the speakers. Those elements are especially important in order to understand debates and their dynamic properly.

A speaker segmentation algorithm using Sentence Embedding and BiLSTM, is proposed. We will execute a pre-trained sentence embedding and tag the sentences using BiLSTM network and Multi-Margin-Loss, which are trained on tagged data.

2.Problem Definition

Speaker recognition, which includes speaker identification and verification, are widely researched in recent years. In these existing speaker recognition systems, it is supposed that the input speech belongs to one of the known speakers. In many applications, it is necessary to find speaker change points first before the speaker can be identified. This procedure is called speaker segmentation, or speaker change detection. Speaker tracking or segmenting a speech stream by speaker identities is essential in many applications, such as conference and meeting indexing, audio/video retrieval or browsing, speaker adaptation for speech recognition and video content analysis, and speaker change detection is a preliminary processing for speaker tracking.

We will focus in this project on examining the ability of a sentence embedding model to be used as an input to a neural network model for tagging sequences of sentences effectively.

The objective of the task we examine in this part of the project is to detect the points where the speaker is changed, i.e., points where the speaker is different from the speaker of the previous sentence, given a transcription of a US presidential debate. We are assuming that we have a given algorithm that divides the debate to sentences, and thus, each sentence can be labeled as a sentence where the speaker has changed or not changed. Our task will be classifying the sentences given the debate segmented by sentences, such that it will minimize the risk. In our experimentation we tested the effectiveness of the combination of the pre-trained sentence embedding with our model for debates speaker segmentation.

3. Data:

3.1. Data Import

We found transcripts for 46 United States presidential debates which includes all the speakers in each debate and everything they said in chronological order in each debate. We imported the data from https://www.kaggle.com/bahdahshin/presidential-debates. This includes the transcripts for all the presidential debates and vice-presidential debates that has ever been. In addition, we set a party to each speaker that is a candidate using information from Wikipedia (necessary for the advanced procedure).

3.2. Data Understanding

In order to create the data that will best fit the models and the problem we tried to solve; we first analyzed the Input we imported. The properties of the data and the problem led us to define two classes of sentences: speaker changed or no not changed.

When we segmented every debate into sentences, we found out that each of the debates includes approximately 900 sentences, and 42901 sentences overall. In approximately 19% of the sentences the speaker changed, and in approximately 81 % the speaker did not change. In addition, 41% of the sentences were said by the republican party candidates, 36% of the sentences were said by the democrat party candidates and 23% of the sentences were said by others such as the moderators, independent candidates and audience. The difference between the republican and democrat candidates' number of sentences can be partially explained by the fact that there was one debate where the democrat candidate was absent. There were 2 debates with significantly higher "other" proportion of sentences since in those debates there were independent candidate.

There was also big difference in the length of debates. The smallest debate contained only 451 sentences and the biggest contained 2055 sentences. The differences can be partially explained by the duration of the debate. Some might have been shorter than most debates. While the biggest debate includes many interruptions of each of the candidates and many short sentences.

3.3. Data Preparation

Each debate is in a different text file. We divided Those text files into 4 types by the structures of the text file, such as differences in punctuation between the speaker's name and his speech. Because of the differences we processed the data in a slightly different way for each debate (depending on the type), in order to create unified train and test datasets.

For each debate we segmented the entire debate to sentences using "Punkt" sentence tokenizer. We assigned for each sentence the name of its speaker, an indicator which indicates if the speaker has changed. For the advanced procedure, we added a party association label for each sentence where the tags are "democrat", "republican" and "other" where "other" is a speaker who isn't a democrat candidate and isn't a republican candidate.

We used the pre-trained sentence embedding model "paraphrase-mpnet-base-v2 " which converts a sentence into a 384-dimension tensor, and we applied it to each sentence. We prepared the sentences for sentence embedding by removing non-English characters (such as ' — ').

After that preprocess, we split the debates randomly to train and test datasets which include complete debates. The train dataset contains 40 debates and is used also for validation. The test dataset contains 6 debates. We kept the debates complete during the split so that we wouldn't lose information that can be inferred from its complete dynamics, e.g., in some parts of a debate there is a higher probability that a candidate will interrupt the other candidate, and a complete debate implies the positions of its sentences in the debate.

4.Models and Algorithms:

Our basic model is BiLSTM network followed by scorer that consists of a linear layer and a log-soft-max function.
We'll define N = number of sentences in a debate.
The number of classes is 2 when the task is speaker segmentation.

Before we feed the network, we have already embedded all the sentences so that the input to the BiLSTM network is a [1, N, sentence-embedding-dimension] shape tensor. That tensor which represents one debate enters the BiLSTM network. The BiLSTM output is a [2*LSTM hidden state dimension ,1, N] shape, and it's fed to a linear layer and then to a log-soft-max scorer which gives a score for each class and each sentence. The label chosen for prediction for each sentence is the label with the highest score. This means that the output of the inference method for one debate is a sequence of N predictions.

We use Pytorch package for defining this network. This allows us to calculate derivations of functions for the network's parameters.

We chose Multi–Margin-Loss as our loss function. The log-soft–max scores enter the loss function. The function calculates the loss and from there the model calculates the gradients.

The training of the model was done using gradient methods. We used the Adam optimizer and the CosineAnnealingLR scheduler which changes the learning rate in each epoch. Our motivation is to begin with a high learning rate and reduce it during the training.

We used batch size = 1. We chose this size for every batch because we have a relatively small amount of debates but every debate is full of data and we didn't want to split any debate as explained in the previous page.

We used a dropout method for training where we randomly replaced sentence embeddings with tensors which represent unknown sentences. That commonly used method should improve generalization.

In the table below we summarized the hyper parameters we chose where some of them were optimized using cross validation and some of them weren't optimized:

Epochs	30
LSTM_hidden_dimenstion	200
LSTM_num_layers	2
LSTM_dropout	0.2
Dropout of the input	0
Learning rate	0.007
Sentence_embedding_dimension	384

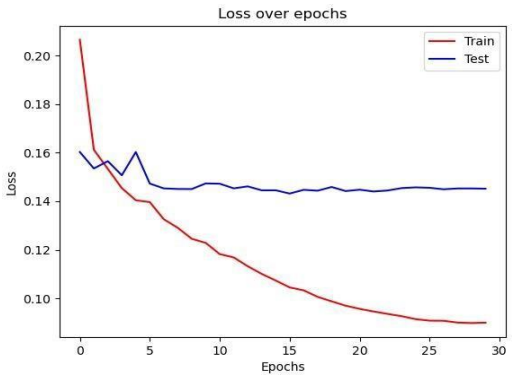
5.Experiments and Results:

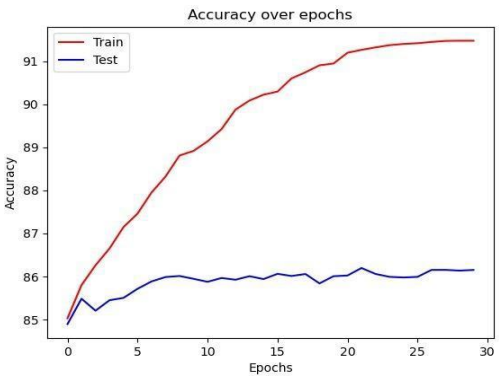
In our experimentation we tested the effectiveness of the combination of the pre-trained sentence embedding with our model for debates speaker segmentation. Throughout the evaluation and analysis process for our models, we used the accuracy metric - defined in this project as the average percentage of correct sentence labeling across the debates (higher is better).

We used 5-fold cross validation and the Optuna package to find the best hyper parameters. The hyper parameters which we tried to optimize are: 'lstm_hidden_dimenstion', 'lstm_n_layers', 'lstm_dropout', 'dropout', 'lr'. The objective for the cross-validation is the mean accuracy of the best epoch accuracies of the 5 trained models.

Finally, we trained the model with the best hyper-parameters, using all the train set. After each epoch of that training, we calculated the train and test total losses, and train and test accuracies. We chose the parameters which maximized the test accuracy. The Table and the graphs below present the results that we got from those parameters and the metrics we measured in each epoch.

Test Accuracy	86.198
Train Accuracy (in best epoch of test)	91.262





We got Test accuracy of over 86%. While it might seem high, remember that the classes in the datasets are imbalanced, i.e. In approximately 19% of the sentences the speaker changed, and in approximately 81 % the speaker not changed. So, high accuracy was somewhat expected

The accuracy on the test set comes close to it's peak after relatively small number of epochs but it still continue to slightly improve until it reaches it's peak in epoch 22 and then it settles.

We believe there are several reasons that we didn't get higher test accuracy. One possible reason is that while sentence embedding keeps many of the information it receives from the text, large part of it is lost. Another possible reason is that in some of the points where the speaker changes, there is no textual cue of the changing, hence, it is very difficult to infer the changing. It's possible that to such sentences the realizability assumption does not hold, and the optimal risk is much higher than 0.

Despite that, the fact the test accuracy is higher than the proportion of the majority class shows that the model manages to learn and that the sentence embeddings does retain relevant information for the task.

6.Advanced Part

In this part we would like to examine how much the labels of "speaker changed" can help to a model like our basic model to classify debate sentences by party of the speaker. We modified our model so that we will classify the party of the speaker in each sentence.

The model here is also a sentence embedding that enters a BiLSTM network and a scorer which gives a score for each sentence and each party.

In addition, we looked at a model that is like the model described above, except that it receives a sentence embeddings tensor and another tensor that is made of one-hot vector of the "speaker changed" label of the sentences as inputs. That means that for every sentence in the debate, the model receives an input composed of a sentence embedding vector, and another input vector which is (0,1) if the speaker has changed, and (1,0) if the speaker has not changed (and (0,0) if unknown). The two vectors of each sentence are concatenated and fed to the LSTM network.

We were expecting that the addition of "speaker changed" data would improve the accuracy of the model because usually "speaker changed" usually comes along with a change in the party of the speaker.

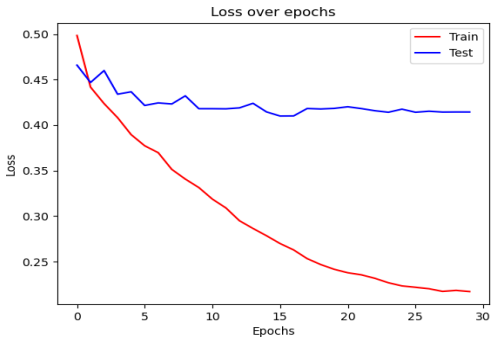
In the tables below we summarized the hyper parameters we chose where some of them were optimized using cross validation and some of them weren't optimized:

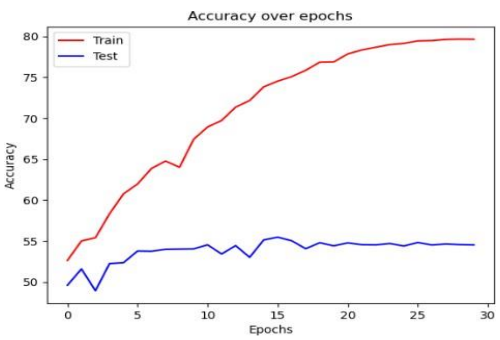
	Only Sentence Embedding Input	Sentence Embeddings + "speaker changed" Input
Epochs	30	30
LSTM_hidden_dimenstion	251	335
LSTM_num_layers	3	2
LSTM_dropout	0.194	0.03
Dropout of the input	0.064	0.13
Learning rate	0.0028	0.0044
Sentence_embedding_dimension	384	384

We compared the results of the accuracies of the 2 models, and we saw the following results:

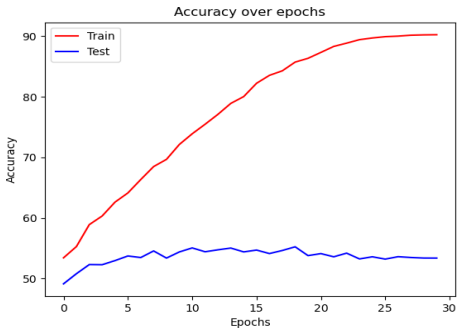
	Test Accuracy	Train Accuracy
Only Sentence Embedding Input	55.477	74.531
Sentence Embeddings + "speaker changed" Input	55.232	85.729

The graphs for the model with only sentence embedding input:





The graphs for the advanced model with Sentence Embeddings + "speaker changed" Input:



It can be seen from the table and the graphs that the results for both models yielded very similar results. It means that the advanced model with the "speaker changed" input didn't manage to exploit all the new given information. We assume that the issues that caused this are related to our data (and not to our model), e.g., the dynamics of speaker changes are indifferentiable between candidates of different parties.

7.Creative Part

In this part we would like to optimize a different objective for a speaker segmentation. We would like to maximize the F1 score of the "speaker changed" class (the minority class). The F1 score is defined as: $F_1 := 2 * \frac{Precision * recall}{precision + recall}$.

We used the average precision and average recall of "speaker changed" sentence tagging across the debates, instead of precision and recall variables of the formula. We used the "F1_Loss" loss function that is dedicated for optimizing macro-F1 score. It was based on the work published by Michal Haltuf on Kaggle. That loss gives for each sentence a probability for each class using the soft-max scores that are received from the model and calculates macro–F1 score with the continuous probabilities instead of binary predictions. Therefore, the loss is differentiable and suitable as a loss function. In addition, we defined another loss function that we called "speaker changed F1 loss". This loss calculates the (binary) F1 score of the speaker changed class only instead of calculating macro-F1 score like "F1_Loss" does. We trained a model using "F1_Loss" and compared the F1 test result with the F1 test result of 2 other models: 1. the basic model 2. a model trained with "speaker changed F1 loss".

We were expecting to receive higher F1 score for a model trained with "speaker changed F1 loss" rather than the F1 score for the basic model, since it uses a loss function that is dedicated for optimizing F1 score of "speaker changed" class. Furthermore, we were expecting to receive for a model trained with "F1_Loss" almost as high F1 score as the score of a model trained with "speaker changed F1 loss", because it optimizes a similar metric.

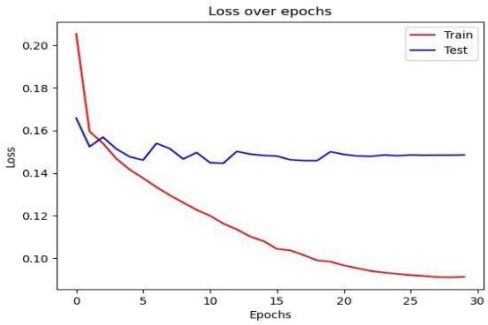
In the tables below we summarized the hyper parameters we chose where some of them were optimized using cross validation and some of them weren't optimized:

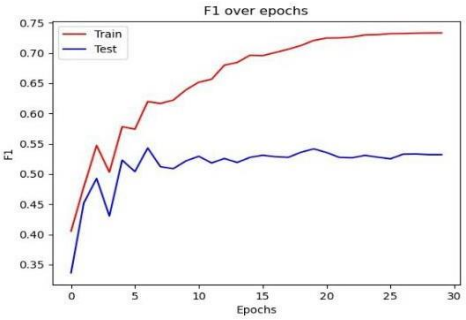
	Basic Model	"F1_Loss" model	"speaker changed F1 loss" model
Epochs	30	30	30
LSTM_hidden_dimenstion	200	80	80
LSTM_num_layers	2	2	2
LSTM_dropout	0.2	0.2	0.2
Dropout of the input	0	0.11	0.11
Learning rate	0.007	0.008	0.008
Sentence_embedding_dimension	384	384	384

We compared the results of the F1 scores of the 3 models and we saw the following results:

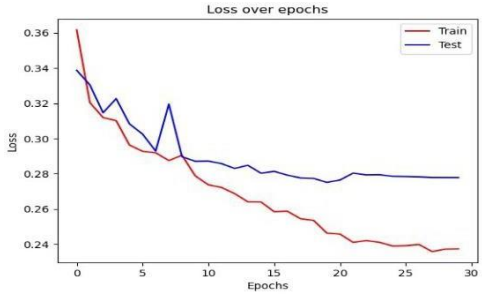
	Test F1 score	Train F1 score	Precision	Recall
Basic Model	0.543	0.620	0.648	0.467
"F1_Loss" model	0.552	0.641	0.585	0.522
"speaker changed F1 loss" model	0.546	0.665	0.547	0.546

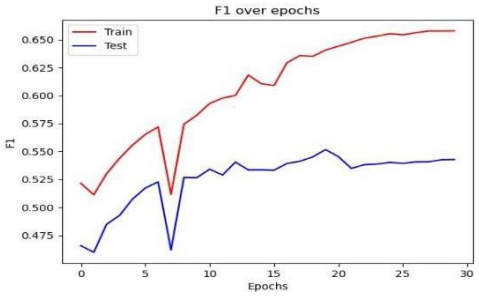
The graphs for the basic model results:



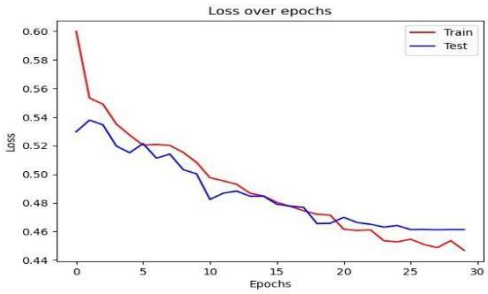


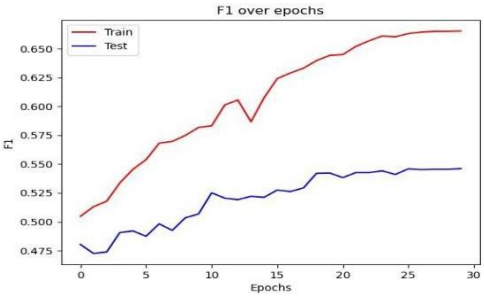
The graphs for the "F1_Loss" model results:





The graphs for the "speaker changed F1 loss" model results:





It can be seen from the table that our "F1_Loss" model yielded similar but slightly better results than the basic model and slightly better results than the "speaker changed F1 loss" model. We notice that the basic model yielded a higher average precision score than the other models, while the basic model yielded a lower average recall then the other models. A possible explanation is that the basic model maximizes accuracy, and the classes are imbalanced. Therefore, the basic model is biased to tag sentences as "not changed" and that increases precision and reduces recall. On the other hand, the other models maximize the harmonic mean of their precision and recall, which gives a high penalty when one of the metrics (precision, recall) is low. From that we infer that our creative models are better then the basic model in detecting speaker changes.

Although we received a higher result in the "F1_Loss" model, we didn't test statistical significance. Future research is required for confirming the results.