Generating Smart Minutes of Meeting using BERTSum model for extractive summarization

Amit Garg
San Jose State University
amit.garg@sjsu.edu,

Maaz Sirkhot
San Jose State University
maazsadique.sirkhot@sjsu.edu

Shriya Vanvari
San Jose State University
shriya.vanvari@sjsu.edu

Shubham Kumar
San Jose State University
shubham.kumar@sjsu.edu

Abstract

Our model helps a user go through minutes of meeting in such a way that user gets meeting summary generated by a tool that picks up only relevant information to agenda of meeting and discards any unnecessary information. As a part of the solution, an application is to be designed which takes audio recording of the meeting and generates the minutes of the meeting and also analyses the sentiment of the meeting. Instead of a person spending time on taking the minutes and causing a scope of human error, a system to auto generate the minutes can be developed. An interactive UI developed using ReactJs which allows a user to either record meetings in real time or upload a transcript of a past meeting is designed for the ease of the customer where he can obtain the minutes of his past meeting as well.

Keywords: NLP, Extractive Summarization, Sentiment Analysis, Named Entity Recognition,Machine Learning

1 Introduction

In today's era, Artificial Intelligence is being used to reduce human efforts as much as possible by leveraging machine learning and statistical techniques and high performance computing. Neural networks have transformed the artificial intelligence industry by providing novel techniques that can be used in every possible domain just by tweaking the architecture and providing enough data. The scope of applications is humongous. One such application is text summarization. By leveraging this technique, we have come up with a model that with proper processing generated the minutes of a meeting effectively by using extractive summarization technique.

A transcript is generated from real time audio recording. This transcript is then provided to Grammar model which adds punctuation and corrects the grammatical mistakes in the transcript for a better summarization. The punctuated transcript is then provided to summarizer model which returns the minutes of the meeting. The transcript is also passed on to the sentiment analysis and Named Entity Recognition model where the names in the transcripts are extracted so that emails can be sent to the said people.

2 ML Techniques

Different ML techniques have been used in our system to generate desired output. There are four ML modules in our system namely Grammar model, Summarizer model, Sentiment Analysis model and Named Entity Recognition model.

2.1 Grammar model

To validate and correct the grammar in the transcript, we used Deep Text Corrector model.Deep Text Corrector uses TensorFlow to train sequence-to-sequence models that are capable of automatically correcting small grammatical errors in conversational written English. It does this by taking English text samples that are known to be mostly grammatically correct and randomly introducing a handful of small grammatical errors (e.g. removing articles) to each sentence to produce input-output pairs (where the output is the original sample), which are then used to train a sequence-to-sequence model. The data set used to train this model is the Cornell Movie-Dialogs Corpus, which contains over 300k lines from movie scripts. A sample sentence is drawn from the dataset. The input sequence will be obtained by randomly applying certain perturbations on this sentence. Output sequence will be the unperturbed sentence where the perturbations applied are intended to introduce small grammatical errors which we would like the model to learn to correct.

Instead of using the most probable decoding according to the seq2seq model, this model takes advantage of the unique structure of the problem to impose the prior that all tokens in a decoded sequence should either exist in the input sequence or belong to a set of "corrective" tokens. The "corrective" token set is constructed during training and contains all tokens seen in the target, but not the source, for at least one sample in the training set. The intuition here is that the errors seen during training involve the misuse of a relatively small vocabulary of common words (e.g. "the", "an", "their") and that the model should only be allowed to perform corrections in this domain.

2.2 Summarization model

Text summarization can be of two types: Extractive and Abstractive Summarization. While Extractive Summarization returns the most impactful and insightful sentences from the input text itself, abstractive summarization generates completely new sentences. We tried out PreSumm model for Abstractive summarization in the past and realized that the sentences returned did not make sense at all and also were quite irrelevant to tha actual transcript not providing any useful information at all. Hence, we fine tuned BERT for extractive text summarization (BERTSum).

The format of BertSum's input is slightly different compared to original BERT. In order to represent individual sentences, [CLS] tokens are inserted at the start of each sentence, and each [CLS] symbol collects features for the sentence preceding it. Also,it uses interval segment embedding to distinguish multiple sentences within a document. For segment embedding EA or EB is assigned depending on whether sentence is odd or even. For example, for document [sent1, sent2, sent3, sent4, sent5], would assign embedding [EA, EB, EA, EB, EA]. This way, document representations are learned hierarchically where lower Transformer layers represent adjacent sentences, while higher layers, in combination with self-attention, represent multi-sentence discourse.

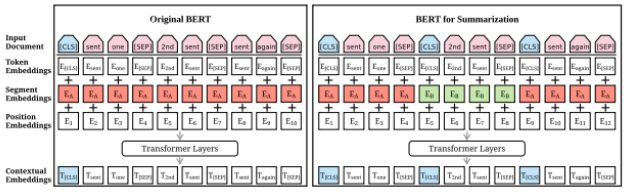

Figure 1: Difference between BERT and BERTSum Input

For training target, BertSum labels representative sentence, documents [sent1, sent2, sent3, sent4, sent5] had label [label1, label2, label3, label4, label5]

BertSum outputs a list of scores shows the representativeness of sentence toward documents. For documents [sent1, sent2, sent3, sent4, sent5] has scores [score1, score2, score3, score4, score5]. The higher score is, the more representative the sentence is. The output of Bert is then feed into Summarization Layers for summarization. Refer Figure 2.

2.3 Named Entity Recognition model

Named Entities are nothing but real world objects with names. We have incorporated this so that in future the scope can be enhanced such that we store the email addresses of ech employee in our system and as soon as the named entity recognition model recognizes that a certain individual was mentioned, it will append it to the named entities list and upon completion it will ask user if they want to send out the minutes too all the detected named

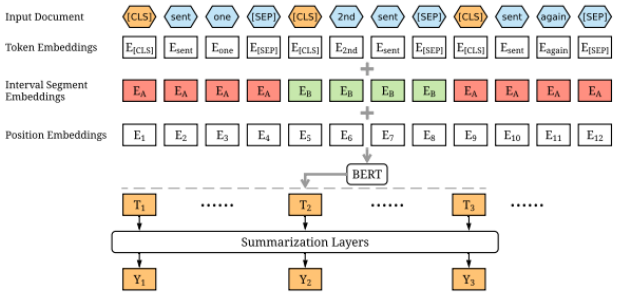

Figure 2: BERTSum Output

entities. We have used Spacy library for Named Entity Recognition in the transcript. Spacy is a python NLP framework. Spacy uses Convolutional Neural Network for Named Entity Recognition.

Other framework that provides NER is sci-kit. However, Spacy provides a lot of flexibility and NLP techniques to pre-process the data for stemming, parts of speech tagging, etc. Spacy can also be configured for cutom named entity recognition which would be helpful for further enhancements of the project.

2.4 Sentiment Analysis model

Our system detects the sentiment of the transcript using Support Vector Machine(SVM). SVM is a supervised machine learning technique that is majorly used for classification. SVM classifies the data points by creating a hyper plane between the classes in N dimensional space where N is the number of classes. In our case, the number of classes would be three i.e. Positive, Negative and Neutral. SVM draws the hyperplane by transorming the data into mathematical functions using Kernels. SVM can use many types of Kernels ie Linear, RBF,sigmoid, etc.

Since in our case the classes are linearly separable, we can use Linear kernel for the classfication of the sentiment. . However, we do not provide the text directly to the SVM classifier. The transcript is first vectorized using Tf-Idf vectorization which helps us to get word embedding vectors. These vectors are then provided to our linear SVM which classifies the text as positive negative or neutral. We got a f-1 score of 91% which is the arithmetic mean of precision and recall. Hence we got a decent accuracy on the classification of sentiemnt of the meeting.

3 System Architecture

We have implemented 3-tier architecture involving Frontend server, Backend Servers and database. A request is created for each activity to be performed or any computation is required. This request is passed on to APIs residing on backend servers. These APIs managed by NodeJS, process requests GET, POST, PUT, DELETE

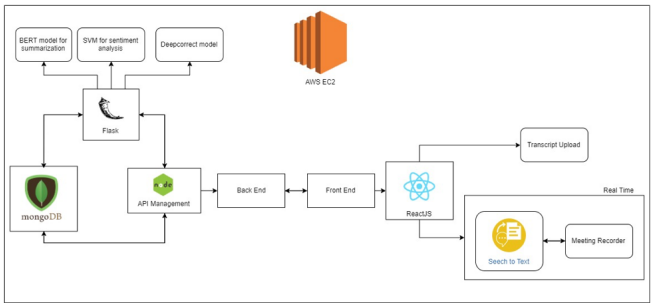

Figure 3: Architecture Overview

which are responsible for validating the request and body parameters. Business logic and computation is performed by these APIs that follow single responsibility model where each API performs one function. These APIs interact with NoSQL database MongoDB using Mongoose module. All the data including profile data, meeting transcripts, summaries are stored in database.

Apart from NodeJS backend, there also is another set of APIs which act as services for NodeJS APIs. These APIs are written in Python's Flask which mainly perform the task of generating summary for meeting transcript. When a request is received to generate summary for a transcript, the transcript is stored in database and the reference for the same is passed as a parameter to Flask API service. Using this reference, it retrieves the transcript text from database and generates summary using various NLP models. Once a summary is generated, it is stored in database and its reference is passed as a response back to NodeJS API. This API fetches the summary from database and provides it to the frontend to further display it on web interface.

3.1 Frontend

Frontend controls the appearance, behavior and input validations for entities. Developed using ReactJS, web interface is a component-based model where each component acts independently and communicate with each other with the help of props passed from parent to child component. There are two major ways of acquiring smart summaries for transcripts.

• One is to upload a text file which contains the transcript of the already occurred meeting. A user provides the text file along with meeting agenda and lines of summary needed. • Other is a breakthrough real-time speech to text feature which allows users to record the audio during a meeting real time and can see the text on application. Once the meeting is completed, user can stop the recorder, review the transcript and upload it along with the agenda and lines of summary. This functionality is implemented using Web Speech API in React.

Although, user is required to provide microphone permissions to the browser in order to use this feature seamlessly.

The summary is promptly displayed on the web page in a jumbotron component which appears only when a summary is supposed to be displayed. This includes not just the summary and topic name but also the list of entities mentioned in the transcript as well as the sentiment of the meeting.

These features are available exclusively to the users who have signed up with a unique username/email and password. Other details such as name and country are obtained from the user. The web interface also provides public access to analytics metrics such as the number of users signed up, number of countries from which various users have signed up, the number of meetings for which summary has been generated and also the number of meeting duration minutes recorded using our in-built recorder. These metrics can be useful for gaining various insights as well as allure more users to understand the usability of our application and sign up to generate smart minutes of meeting for themselves. The signed-up users also will be able to update their profile information, upload a profile picture and retrieve all their past summaries generated. This allows the user to keep a track of all the meetings for which they have generated the summary. Hence, they don't have to generate summary again for the same transcript in case they to send the minutes of meeting to someone else later.

3.2 Backend

API Management: The backend server consists of all the APIs designed to perform various computation and business logic processing for the requests that are received from client. This is the backbone of the application is responsible for accurateness and robustness of the application. With the REST APIs design, it enables horizontal scalability and fault tolerant architecture. The APIs accept request data in JSON as well provide JSON responses along with status codes for the client to easily classify the responses and take appropriate actions. The server makes sure proper status codes are sent for all possible cases to prevent application from crashing. Some of the status codes used are 200, 400, 404, 501, 304, 204. Concrete exception handling and edge case management are key towards preventing unintended crashes and request failures.

Password Encryption: With the security being the utmost priority, the user account passwords are not stored in plaintext which is a high-risk method prone to myriad of attacks and data compromise. Therefore, Bcrypt password encryption which is a module in NPM is used to hash the password before storing it in the database. The passwords are hashed with 10 salt rounds making it almost impossible to crack using a brute force attack. Such a robust password encryption and OAuth management using JWT strategies is most critical security measure to secure critical information.

Python APIs: Two APIs were written in Python Flask. All the APIs take the reference of the transcript-sent by NodeJS API as the parameter.One is the summarize API which first corrects the grammar of the transcript using DeepCorrect and then summarizes the gen-

erated grammatically correct text using BERTSUm and stores the summary and the sentiment of the transcript recognized by the TextBlob model in the database and passes the reference of the transcript to the NodeJS API. The other is the entity API which recognizes entities using the Spacy model and stores a list of entities in the given transcript in the database. **Image Upload using Multer:** The image upload for profile picture is a key feature and very important to provide a personalized experience to the users. It enhances the user experience significantly and hence widely used. In this application, Multer module of NPM is utilized to create static webpages for displaying images stored on backend server. These images are references on web interface to display the images. This is a scalable feature and in case the images take up more memory on Backend server, they can also be uploaded on Amazon's S3 bucket using Multer and similar procedure can be followed.

3.3 Database

For storing all the information, NoSQL database MongoDB is implemented as it provides dynamic field creation and object like storage of data. With the flexibility and quicker retrieval time provided MongoDB, it is convenient to store arrays, nested objects and custom datatypes which cannot be done in relational databases. MongoDB Atlas provides a cloud-based solution for storing data efficiently and also provides three clusters by default to provide sharding and connection management. The connection pooling mechanism works seamlessly in MongoDB allowing quicker query turnaround time by dynamically creating connections threads requested by the backend server. For this application, there are three collections namely Users, Transcripts and Summaries which store the respective data. To connect this database with APIs, Mongoose ORM of NPM module has been utilized which provides systematic query creation, prevent unauthorized access and schema-based insertion. Implementing an ORM to access database is more readable and makes the database interaction standardized across the application.

4 Results

Below are the results received by our model on a conversation data. As the length of the summary provided by the user was 5,the 5 most insightful sentences from the conversation were extracted.

5 Further Enhancements

We have planned to broaden the scope of our project by:

1. Implementing a model that will automatically generate a list of tasks assigned to the employees and send an automated email to the said employees.
2. Include facial emotion recognition to get better insights of overall sentiment during the meeting.
3. Provide our model to the consumer as a WebX plugin or an Alexa skill

6 Conclusion

Our model provides an ease of taking effective minutes of meeting to our customers ie corporate employees.

7 Acknowledgement

We would like to thank Professor Ranjan for all the feedbacks provided to enhance our system and also to encourage us to think in a "Design Thinking" manner to better suit our customer's needs. This project really helped in enriching our knowledge in NLP and full stack development domain.

References

Yang Liu,Mirella Lapata *Text Summarization with Pretrained Encoders.* arxiv :1908.08345,5th September 2019

Yang Liu *Fine-tune BERT for Extractive Summarization* arxiv :1903.10318,5th September 2019

Neil Wu: A Bert based summarization model "BertSum" `https://medium.com/lsc-psd/a-bert-based-summarization-model-bertsum-88b1fc1b3177`

Deep Text Corrector Github `https://github.com/atpaino/deep-text-corrector`