CSE 538 - Project Report

Aditya Karamchandani
112950866
akaramchanda
@cs.stonybrook.edu

Pulkit Dongle
112504649
pdongle
@cs.stonybrook.edu

Abstract

In recent years, there has been a explosion in the amount of text data from a variety of sources. This volume of text is an invaluable source of information and knowledge which needs to be effectively summarized to be useful. In this application, we are using a text summarization model to extract the summary of the comments written by students for the different courses on the Stony Brook University's classie-evals website. We also used our own metric and built a model to predict the rating of the courses based on the summary obtained. The implementation is publicly available on GitHub.

1 Introduction

With this application we are trying to build a system where the incoming students get help in deciding which courses they should take based on the comments of the students who have taken the course already in previous offerings of the course. Instead of going through all the long comments of the students on the classie-evals website, they can read a short summary of what are the main highlights of the course and what are some of the things that could have been improved about the course. With this information, the new students can make an informed decision and choose their courses accordingly. To measure the performance of our system, we will be implementing a model which will predict the rating of the course based on the extracted summary and compare it with the original rating of the course on the website.

Automatic text summarization is very challenging, because when we as humans summarize a piece of text, we usually read it entirely to develop our understanding, and then write a summary highlighting its main points. Since computers lack human knowledge and language capability, it makes automatic text summarization a very difficult and non-trivial task.

The summarization model could be of two types:

- Extractive Summarization: Is akin to using a highlighter. We select sub segments of text from the original text that would create a good summary.
- Abstractive Summarization: Is akin to writing with a pen. Summary is used to extract the gist and could use words not in the original text.

We will be using Extractive Summarization in our application. For achieving summarization, there are preexisting models which we will be using and fine tuning it according to our requirements.

(Nallapati et al., 2016) describes the Abstractive Text Summarization using Sequence-to-Sequence RNNs and Beyond. This paper implements an Attentional Encoder-Decoder Recurrent Neural Network to summarize the text and is able to achieve state-of-the-art-performance on Gigaword Corpus and DUC Corpus.

(Liu, 2019) describes a simple variation of BERT known as BERTSUM for extractive summarization. This system has achieved state-of-the-art-performance on the CNN/Dailymail dataset.

(Lin, 2004) For evaluating the summarization models, there are automatic evaluation methods which are a set of metrics since the early 2000s. ROUGE is the most widely used metric for automatic evaluation.

Recall-Oriented Understudy of Gisting Evaluation (ROUGE) is a method to automatically determine the quality of a summary by comparing it to another set of (ideal) summaries often created by humans.

We defined our own metric to evaluate the summaries. After we successfully summarized the student comments, we built a model that will predict the rating of course (say 1 to 10 with 10 being the best) based on both the positive and negative comments and compared with the original mean rating given by the students for the course on classie-evals. We are defining this as a metric for evaluating how good the generated summaries are.

2 Summarization Model Pipeline

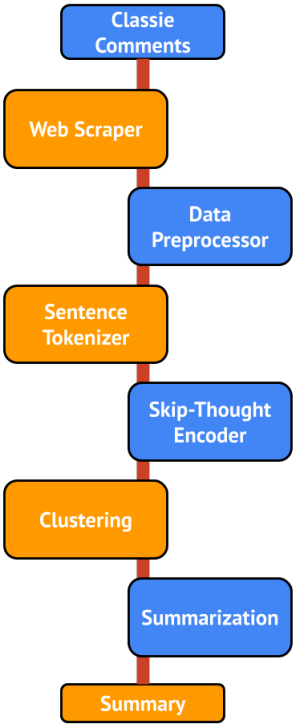

Figure 1: Model Pipeline

2.1 Web Scraping

To obtain the training data, we have scraped the classie-evals website to obtain a total of 15,000 courses offered by different departments at Stony Brook University. We obtained all the information that is available for that course from the classie-evals web-page including the course name, professor's name, grades that students got, the grade that students gave to the course, comments of students, semester in which the course was offered, etc. We would like to thank David Graff (CS Senior at Stony Brook University) for this.

2.2 Data Cleaning and Preprocessing

From the data scraped, we obtained the comments of students who have previously taken the class. The comments were taken from answers of the two sections, "What was valuable about this course?" and "What could be improved about this course?". The comments from the first question were labeled as the positive and the comments from the second question were labeled negative for the purpose of training. While selecting the courses on which we want to train the data, we decided to take the courses where the either the length of positive or negative comments were greater than 500 characters. The different comments from different students were combined into one big paragraph.

2.3 Sentence Tokenization

Once we have obtained all the data, we used NLTK's sentence tokenizer to split each set of comments into its constituent sentence using specific set of rules for sentence delimiters for the English language. Figure 1 shows how sentence tokenizer works. This is done to get the sentence representations from the Skip-Thought model.

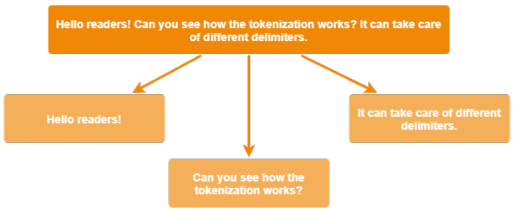

Figure 2: NLTK's Sentence Tokenizer

2.4 Skip-Thought Encoder

We wanted a way to generate a fixed length vector representations for each sentence in our comments from each course. These representations should encode the inherent semantics and the meaning of the corresponding sentence. We used Skip-Gram Word2Vec (Mikolov et al., 2013) method for generating word embeddings for the individual words that are present in our model's vocabulary.

The simple forms of sentence embeddings we saw in lectures were the Deep Averaging network (DAN) (Iyyer et al., 2015) and a model based on GRU-RNN (Chung et al., 2014). The DAN model is a very simplistic way of obtaining sentence embeddings by averaging out the word embedding vectors. This approach as we know is loses the sequential structure of the input sentence, which is important in most languages, including English. So we made a GRU-RNN based model as a part of our assignments as well. Going on further

from these ideas and taking inspiration from (Arora et al., 2019) we planned on using the Skip-Thought model to represent individual sentences into a fixed size vector embeddings.

Choosing such a model for obtaining sentence embeddings overcomes the primary issue of sequence of words not being taken into account. Simpler methods are prone to lose or not encode the information provided in the input sentence into the sentence embedding. Skip-Thoughts as it uses a GRU-RNN Encoder-Decoder architecture can take the sentence structure and the sequence of words into account and generate far richer and information dense embeddings. Following are the components of the Skip-Thought model described in more detail:

1. **Encoder Network:** The encoder is typically a GRU-RNN which generates a fixed length vector representation $h(i)$ for each sentence $S(i)$ in the input.
Let $w_1, \cdots, w_i^N$ be the words in sentence s_i where N is the number of words in the sentence. At each time step, the encoder produces a hidden state h_i^t which can be interpreted as the representation of the sentence $w_1, \cdots, w_i^t$. The hidden state h_i^N thus represents the full sentence. To encode a sentence, we iterate the following sequence of equations as described in (Kiros et al., 2015)

$$r^t = \sigma(W_r x^t + U_r h^{t-1})$$
$$z^t = \sigma(W_z x^t + U_z h^{t-1})$$
$$\bar{h}^t = \tanh(W x^t + U(r^t \odot h^{t-1}))$$
$$h^t = (1 - z^t) \odot h^{t-1} + z^t \odot \bar{h}^t$$

where $\bar{h}^t$ is the proposed state update at time t, z^t is the update gate, r_t is the reset gate ($\odot$) denotes a component-wise product. Both update gates takes values between zero and one.

2. **Decoder Network:** The decoder network takes this vector representation $h(i)$ as input and tries to generate two sentences $s(i-1)$ and $s(i+1)$, which could occur before and after the input sentence respectively. Separate decoders are implemented for generation of previous and next sentences, both being GRU-RNNs. The vector representation $h(i)$ acts as the initial hidden state for the GRUs of the decoder networks. Decoding involves iterating through the following sequence of equations as described in (Kiros et al., 2015).

$$r^t = \sigma(W_r^d x^{t-1} + U_r^d h^{t-1} + C_r h_i)$$
$$z^t = \sigma(W_z^d x^{t-1} + U_z^d h^{t-1} + C_z h_i)$$
$$\bar{h}^t = \tanh(W^d x^{t-1} + U^d(r^t \odot h^{t-1}) + C h_i)$$
$$h_{i+1}^t = (1 - z^t) \odot h^{t-1} + z^t \odot \bar{h}^t$$

where, C_z, C_r and C are used to bias the update gate, reset gate and hidden state computation by the sentence vector, and h_{i+1}^t is the hidden state of the decoder at time t.
Given, h_{i+1}^t, the probability of word w_{i+1}^t given the previous $t-1$ words and the encoder vector is

$$P(w_{i+1}^t)|w_{i+1}^{<t} h_i) \propto exp(v_{w_{i+1}^t} h_{i+1}^t)$$

where $v_{w_{i+1}^t}$ denotes the row of V (Vocabulary matrix) corresponding to the word of w_{i+1}^t. An analogous computation is performed for the previous sentence s_{i-1}.

3. **Objective Function:** Given a tuple (s_{i-1}, s_i, s_{i+1}), the objective optimized is the sum of the log-probabilities for the forward and backward sentences conditioned on the encoder representation:

$$\Sigma_t log P(w_{i+1}^t | w_{i+1}^{<t}, h_i) + \Sigma_t log P(w_{i-1}^t | w_{i-1}^{<t}, h_i)$$

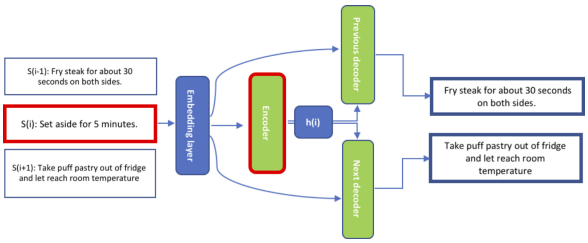

Figure 3: Skip-Thoughts Model Overview

Given a dataset containing a sequence of sentences, the decoder is expected to generate the previous and next sentences, word by word. The encoder-decoder network is trained to minimize the sentence reconstruction loss, and in doing so,

the encoder learns to generate vector representations which encode enough information for the decoder, so that it can generate neighboring sentences. These learned representations are such that embeddings of semantically similar sentences are closer to each other in vector space, and therefore are suitable for clustering. The sentences in the comments are given as input to the encoder network to obtain the desired vector representations. For implementation, we have used the open-source code by the author of the skip-thoughts paper. It is written in Theano and can be found here.

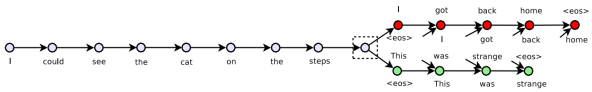

Figure 4: Given a sentence (the grey dots), the model attempts to predict the preceding sentence (red dots) and the next sentence (green dots). Source: (Arora et al., 2019)

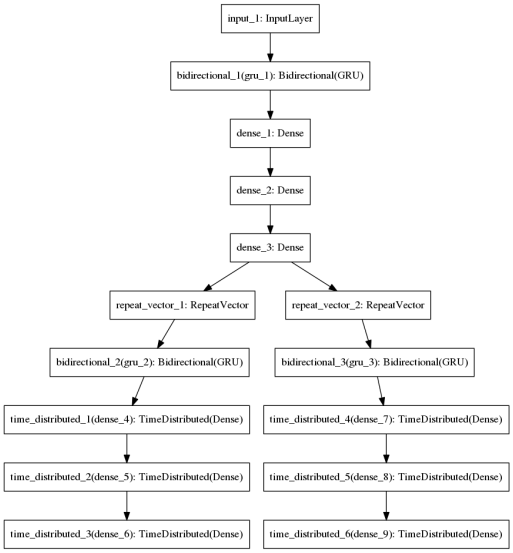

Figure 5: Skip-Thoughts Encoder-Decoder Architecture

The division that you see in the above architecture after the 3 dense layers is because the model attempts to predict the preceding sentence and the next sentence.

2.5 Clustering

Once we are done generating the sentence embeddings for each sentence in a comment, the approach is to cluster these embeddings in high-dimensional vector space into a pre-defined number of clusters. The number of clusters will be equal to desired number of sentences in the summary. For example, if we want our summary to have 6 sentences, then the number of clusters will be 6, one for each sentence. We chose the number of sentences in the summary to be equal to the square root of the total number of sentence in the comment.

2.6 Summarization

We can interpret each cluster of the sentence embeddings as a set of semantically similar sentences whose meaning can be expressed by just one candidate sentence in the summary. The candidate sentence is chosen to be the sentence whose vector representation is closest to the cluster center. Candidate sentences corresponding to each cluster are then ordered to form a summary for a comment (positive/negative comment by students for a particular course). We have combined all the positive/negative comments by different students as a one single comment by concatenating all the comments. The order of the candidate sentences in the summary is determined by the positions of the sentences in their corresponding clusters in the original comment. For example, a candidate sentence is chosen as the first line in the summary if most of the sentences that lie in its cluster occur at the beginning of the comment.

As this method essentially extracts some candidate sentences from the text to form a summary, it is known as **Extractive Summarization**.

Sample summary:

CSE-538 (Fall 2018) Positive Summary:

"The assignments were heavy, but again, very well curated for learning. This course does a good job of solidifying your basics of NLP. The course content is really nice and updated. Great learning experience, we got a change to read and implement very recent research of NLP. The assignments were very nicely designed and were interesting. I found it interesting to learn the maths behind all the NLP concepts. The take home midterm is a great way of learning. Take home exam could possibly be a little less open ended and subjective. The instructor is very focused and supportive, assignments were meaningful, The systematic way in which concepts were covered, from basic to complex."

CSE-538 (Fall 2018) Negative Summary:

"The TA should integrate the requirement into the assignment readme/pdf, like what is expected

from this function. Like NLP, the course was also ambiguous. 3. The assignments are the same every year with solutions floating all over GitHub. More real-time examples can be provided to help us understand the concepts better. Please be clear in the homework instructions. It felt like I was learning tensorflow rather than NLP in the assignments. The assignments can be a bit more clearer. Need not to be programming assignment. The grading could be improved. None!! No improvement required"

3 Scoring the summaries

We scored the summaries as a method to evaluate how good the summarizing model is. This can be done using Sentiment Analysis. The simpler approach or the baseline approach would be to treat the text to be analyzed as a continuous Bag of Words (BOW), where again the word order would be ignored and only the individual words would be used to get the sentiments of a given input. Such a model would be just a statistical approach and not a linguistic approach. To incorporate the understanding of language into the model and we can improve the baseline approach and get promising results. We would be determining the degree to which a given sentence, i.e the summary, is positive or negative, as that would help us in getting a composite score with which we can predict the average grade assigned to the course by the students.

3.1 Making the Bag of Adjectives (BOA)

We used the adjectives from all the sentences, as adjectives are highly informative of positive and negative sentiments. For each comment we discarded punctuations, and tokenized the strings and got rid of the stop words. We then POS-tagged all the words and got the adjectives to form the BOA.

The word clouds fig. 6 and fig. 7 show the adjectives that match the words in the comments with the bag of adjectives that we created. The words such as, "valuable", "interesting", "good", "useful" convey a positive sentiment in the comments whereas the words, "hard", "difficult", "exam", "unfair", "many" are conveying a negative sentiment.

Some of the words such as "good", "helpful" are shown in both the word clouds, this is because of the context in which they are being written. For example, "it would have been helpful if TAs were more responsive on piazza", is showing negative sentiment but the word "helpful" is there in this sentence. Also, some students tend to write positive comments in negative comments section, like, "Everything was good about the course, no improvements required". Such comments introduces errors in training data.

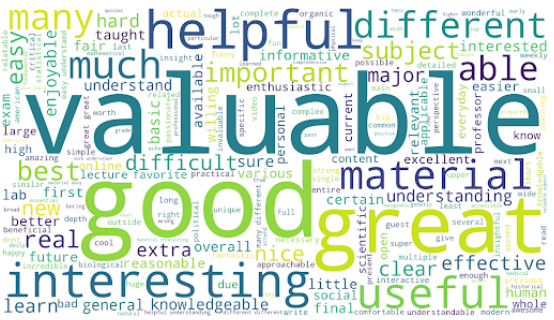

Figure 6: Wordcloud of words with positive likelihood ratio

Figure 7: Wordcloud of words with negative likelihood ratio

3.2 Machine Learning

We used the Naive Bayes classifier as the base model. fig. 8 shows the Normalized confusion matrix of the Naive Bayes classifier.

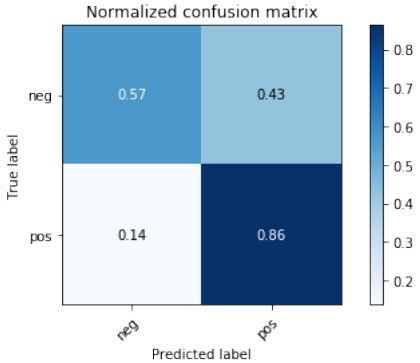

Figure 8: Normalized Confusion Matrix for the Naive Bayes Classifier

Here's an example of the likelihood ratios shown in fig. 9. These ratios depict how likely a word is

to occur in a positive or a negative comment.

```
Most Informative Features
                      be = False           pos : neg    =     38.6 : 1.0
                   fewer = True            neg : pos    =     23.3 : 1.0
          unprofessional = True            neg : pos    =     22.6 : 1.0
              frustrated = True            neg : pos    =     21.2 : 1.0
                    more = False           pos : neg    =     19.6 : 1.0
            inconsistent = True            neg : pos    =     18.4 : 1.0
                  timing = True            neg : pos    =     17.9 : 1.0
                 shorter = True            neg : pos    =     16.5 : 1.0
                  upload = True            neg : pos    =     16.4 : 1.0
                   nicer = True            neg : pos    =     15.8 : 1.0
                    dark = True            neg : pos    =     15.2 : 1.0
                   worse = True            neg : pos    =     14.6 : 1.0
                  nobody = True            neg : pos    =     12.7 : 1.0
                  slower = True            neg : pos    =     12.7 : 1.0
                     rid = True            neg : pos    =     12.6 : 1.0
```

Figure 9: Likelihood ratios of Most Informative Features

We then further tried out all the other classifying algorithms like the Multinomial Naive Bayes, Bernoulli Naive Bayes, Logistic Regression, Stochastic Gradient Descent and Support Vector Classifier. The bar chart in fig. 10 shows the individual accuracies for all these methods.

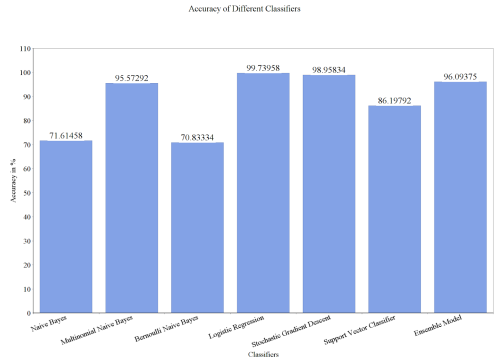

Figure 10: Accuracies different models used for scoring

3.3 Ensemble Model

For the final model, we thought if we could leverage the advantages of all these models and get better results. So we made an ensemble model. This model combined the predictions, i.e. it took votes from all the constituent models mentioned above, for each comment and used the majority of the votes as the final answer. The implementation was easy as it took the mode of the results obtained by the individual classifiers.

To build this ensemble model, we created an EnsembleClassifer class that is initialized with a list of classifiers. The class has two main methods, $classify$: which returns a predicted label (pos/neg) and $confidence$: which returns the degree of confidence in the prediction. This degree is measured as (Number of winning votes)/Total Votes.

4 Results

For evaluating our skip-thoughts summarization model, we first obtained the summary of the course comments from Natural Language Processing (Fall 2018) which was offered by Prof. Niranjan. We pass this summary to our scoring function which provides us the sentiment(positive/negative) and a confidence score. The summaries that we obtained from the skip-thoughts model can be found in Section 2.6

The positive summary got the score of 1 and the negative summary got the score of 0.6. Subtracting the negative score from positive score we get $1 - 0.6 = 0.4$. The grading scale goes from A to D $\{A, A-, B+, B, B-, C+, C, C-, D+, D\}$. If we consider grade D as 0.1, then 0.4 will correspond to grade C on a scale from 0.1 to 1. Thus, based on the comments our scoring function has assigned the course the grade C. The actual grade of the course on classie-evals is $B+$. The reason behind this difference can be many-fold. Some of the reasons can be:

1. We don't know whether the students that are assigning a grade to a course are all writing the comments.

2. The number of positive and negative comments on the course are different which incurs bias.

3. The students that are satisfied with the course may not have written as positive comments about the course as the students who were not satisfied with the course wrote the negative comments.

5 Code

Source Code: GitHub link

1. Original Source Code: Skip Thoughts

2. Files modified

   - All functions of text_summarization.py in the root directory.
   - All functions of scoreSum/classifier.py scoreSum/scoreSum.ipynb

- All functions of scoreSum/train/dataProcessor.py and scoreSum/train/parsejson.py

3. Software requirements

- Python 2.7
- Theano 0.7
- scikit-learn
- NLTK 3
- NumPy latest
- SciPy latest
- statistics python package
- gensim

6 Future Improvements

There are different models available to summarize text. We have used skip-thoughts for the reasons as described in this paper. In future, we can implement different models to compare their performance to see which one produces the best summaries.

Also, for evaluating the summaries, instead of using the NLTK's sentiment analysis package and the sklearn's models, we can create our own Deep Neural Network Model and see whether we get better results.

References

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2019. A simple but tough-to-beat baseline for sentence embeddings. 5th International Conference on Learning Representations, ICLR 2017 ; Conference date: 24-04-2017 Through 26-04-2017.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. In *Association for Computational Linguistics*.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. 2015. Skip-thought vectors. *CoRR*, abs/1506.06726.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Yang Liu. 2019. Fine-tune bert for extractive summarization.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.