Semi-supervised recursive autoencoder for keyword extraction Project Report

Yi Cheng(yicheng1), Anoop Hallur(ahallur), Xiaoqiu Huang(xiaoqiuh)

December 10, 2014

1 Introduction

As the goal of our project, we plan to do keyword extraction using deep learning. We are aiming to improve the performance of keyword extraction by deep learning techniques as compared to other techniques used previously. We believe that Deep Learning techniques can improve the performance because when it has been applied to other similar tasks, a significant improvement has been observed. For example , Google speech recognizer uses Deep Neural Networks and accuracy is extremely high[2].The area of applying deep learning to keyword extraction has not been explored much, hence we are trying to apply it and see how it performs.

2 Problem Definition

We can formally define our problem definition as extracting a keyword from a document using recursive neural networks on a data set of stack exchange posts, and comparing it with the results of baseline algorithms. We also analyze why certain variations of our model resulted in different results and explain why the baseline algorithms give poor results on the chosen dataset.

3 Related Work

Before diving into the details of our model and algorithm, we want to summarize the related work that has been done in the field of Keyword Extraction, how they can be applied to our project and how deep learning can be applied to solve this particular problem and our thoughts on why deep learning should give better performance compared to other algorithms.

3.1 Baseline Algorithms of Keyword extraction

A survey was done by Lott.B[3], and he summarizes that whenever we have a large corpus of data already available(as in our case), TF-IDF is the most accurate algorithms of the existing ones.

In TF-IDF model, we assign weight to each term in the model, and we choose the top'n' weighted words as the keywords for the text. The weight of each word is computed by taking into account the Term Frequency(TF) and Inverse Domain Frequency(IDF), where Term Frequency indicates how significant is the term to a specific document, and IDF takes into account how common the word is in the domain. Other algorithms presently being used are some variant of TF-IDF with domain specific modifications. For example , one technique uses a Bayes classifier with TF-IDF to compute the weights and extract the keywords. It is similar to TF-IDF that takes, but it also takes into account the depth of the word in the document. The intuition is that keywords are present towards the beginning of the document , rather than the towards the end. So we use a third term, called the depth term, in addition to TF and IDF terms. We have implemented the benchmarks for these algorithms on our dataset.

The other algorithm proposed by Lott.B[3] was the one using Lexical Chain approach. In this approach, they extract features from the document, first by constructing a lexical tree of the nouns in the document. They extract the nouns and each noun word is a possible keyword for the document. The lexical chain is constructed using semantic features of the words such as other synonym, hyperonyms and morphynyms of a noun. From the lexical tree, they

extract seven features concerning positions and scores of the nouns. The features are used to train a C4.5 decision tree and this decision tree is used to make predictions of the nouns in the document from the test data. We have implemented this particular algorithm for our data set and analyzed the results.

There are techniques which are to be used when we don't have a corpus of text available to us. Frequency based single Document Keyword Extraction is one such technique. This techniques computes word weights by measuring the frequency of text occurrence within two punctuation marks in the text. Since we have text corpus available, we did not want our algorithm to be compared against these classes of algorithms.

3.2 Deep Learning Intuition

All the machine learning algorithms we talked about in the last section are all shallow-structured architectures. However, these algorithms with simple models and limited representation power cannot achieve impressing results in dealing with complicated tasks like NLP. In order to extract more complex features and better represent human language, multiple layers models or deep structure models are designed to handle more complicated human language problems[7].

The purpose of deep learning is to extract more features from human languages through adopting neural networks with deep structure. So the idea of deep learning originates from artificial neural network(ANN). But unlike ANN, the algorithm of deep learning can not only adopt deep structure but also achieve better optimization results. We will introduce one of the deep structures in the following parts and compare its experimental results with the results of some shallow structures.

4 Model Description

In this part, the neural language model, which is the key part of NLP, will be first introduced. Then the deep learning structure named recursive autoencoder will be discussed in the second part. Finally, we will explain the detail of our model.

4.1 Neural language model

In traditional NLP model, the input of the classifier is just one-hot features or TF-IDF features of text. However, this model suffers from the curse of dimensionality. When the number of documents in the corpus becomes large, the matrix of features will become sparse. Therefore, Bengio et al.[1] proposed a new neural language model which generates distributed representations for each word in the corpus. That means each word in the documents has a fixed length of feature vector. With the feature vectors of each term, we can measure the similarity between different words or generate the vector representation of each document.

In our model, each word will be first transformed to word vector by the tool named word2vec[4]. The tool will first train the neural language model on our data set and then generate the word vector of each term. The structure they used in word2vec is showed in Figure 1. Here we just choose the CBOW model.For each sentence, we pick fixed windows of sentences to train the model. In this model, The input of the neural network is the fixed length vectors of each word in the sentence and these vectors are first randomly initialized. In the hidden layer, these vectors are just summed up and the classifier in the output layer is the hierarchical softmax function. In the training process, each time when a sentence was inputted into the model, all the parameters will be updated. Finally, when all sentences are processed, we can get the word vector of each word.

4.2 Recursive Autoencoder

Autoencoder is a neural network model that learns the function $f(x) = x$ in order to learn the reduced representation of the input text. As we can see in the figure 2, autoencoder is only a three-layer neural network. We want to utilize the hidden layer to generate the similar vector as the input vector. And after training the model, the hidden layer can be treated as the condensed feature of the input vectors. Therefore, the training objective is to minimize the distance which is known as reconstruction error between the input and the output of the Neural Network. When applying such structure into our model, the input of the autoencoder is word vectors of two terms. After traning the model, the vector of hidden layer can be seen as the condensed feature of these two words.

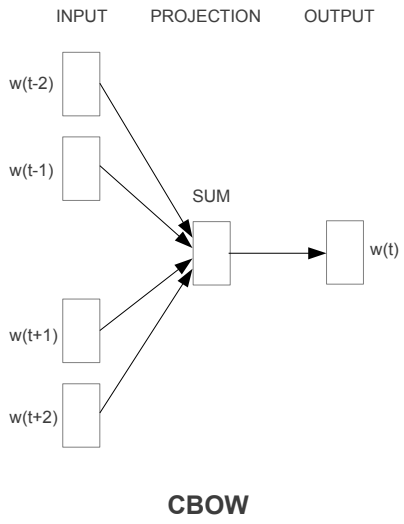

Figure 1: Structure of word2vec model

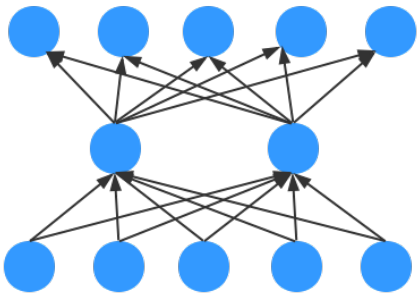

Figure 2: Autoencoder

However, only using the structure of autoencoder may not effectively model the text. If we set the input of autoencoder as the vectors of all terms in one document and try to generate the condensed features of the document, there will be so many parameters to optimize and the structure of the text is not utilized. Besides, Accorrding to Socher et al.[8], the structure of human language is recursive. In order to solve the problem and better represent the human language, the structure of recursive autoencoder was proposed to model the hierarchical structure. Figure 3 shows the structure of recursive autoencoder.

In each layer, several terms are combined to generate the condensed features which will be utilized as the input of the next layer. At the final layer of recursive autoencoder, the vector of hidden layer will be the vector representation of the whole documents.

One of the key problem in such structure is how to combine different terms. There are mainly two ways. One way is to adopt the grammar tree which is generated in advance. Each word will be combined in the same way as they are merged in the grammar tree. Another way is to utilize the greedy algorithm. At each layer, each pair of the adjacent words will be first combined and output reconstruction error. Then the pair of words with smallest error will be first combined. Using this greedy algorithm, the vector of each document can also be generated. In the experimental part, the effectiveness of the two structured will be compared.

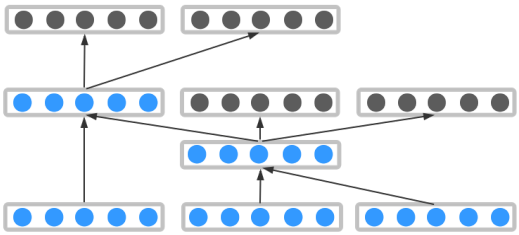

Figure 3: Recursive Autoencoder

4.3 Semi-supervised Recursive Autoencoder

In the traditional recursive autoencoder model, the network is trained only by the document itself. Therefore, it is the unsupervised model. In certain scenario, the label information of documents can be incorporated into the model and turn the model into a semi-supervised model.

One of the semi-superivised recursive autoencoder model has been proposed by socher et al.[9] in order to handle the issue of sentiment analysis. The structure of recursive autoencoder is presented as follows:

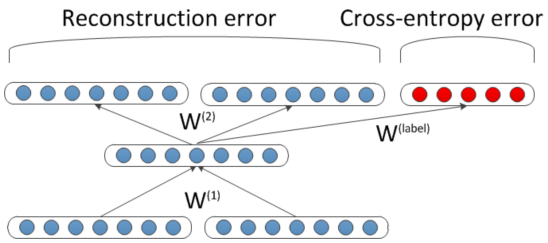

Figure 4: Semi-supervised Autoencoder

As we can see from the figure 4, the input of the model is also the distributed representation. After being processed by the hidden layer, the output layer reconstructs the input vector. In order to measure the performance of the representation, the reconstruction error is computed as the distance between the input vector and output vector. Also we can assign different terms with different weights. So

$$\begin{aligned} &E_{rec}([c_1;c_2];\theta) \\ &= \frac{n_1}{n_1+n_2}||c_1-c_1'||^2 + \frac{n_2}{n_1+n_2}||c_2-c_2'||^2 \end{aligned} \tag{1}$$

, where n denote the number of words. In addition, c denotes the vector of input and c' present the vector of output.

Also, in order to predict the sentiment of the sentence, an extra output unit is added to generate the sentiment label. And this generated label will be compared with the correct label. A softmax function is utilized in this scenario to generate the probability of each label. And author use the cross-entropy error to measure the correctness of the model.

$$\begin{aligned} &E_{cE}(p,t;\theta) \\ &= -\sum_{k=1}^{K} t_k \log d_k(p;\theta) \end{aligned} \tag{2}$$

, where t_k denotes the target label and d_k denotes the probability of each label.

After combining the two process, we can compute the reconstruction error and cross-entropy error. And the objective is to minimize the weighted sum of the two errors as follows:

$$\begin{aligned} &E([c_1;c_2]s, ps, t,) = \\ &= \alpha E_{rec}([c_1;c_2]_s;\theta) + (1-\alpha)E_{cE}(p_s,t;\theta). \end{aligned} \tag{3}$$

4.4 Grammar Tree

As we know, deep learning methods have two the basic recursive structures. One is built by the greedy algorithm. Another is just to merge two input units into one from left to right. Here, we want to replace these structures with the structures of grammar tree of each sentence. We think the structure generated by the grammar tree may be more reliable than the structure generated by previous methods.

In our work, the constituency grammar tree of each document is generated by the stanford parser[6]. Actually, the structure of grammar tree is also generated by a recursive neural network(RNN). The basic structure of this RNN is shown in figure 5.

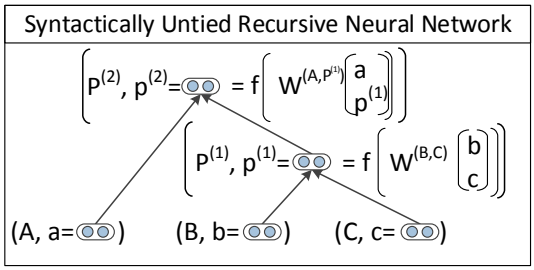

Figure 5: The structure of grammar tree

4.5 The structure of our model

In our model, the semi-supervised structure is applied to handle the issue of keyword extraction. The overall workflow is shown in the Figure 6. After getting the training data, we first preproccess those data. And then each word is transformed to a word vector by word2vec and the stanford parser is utilized to build the grammar tree for each sentence. After that, these two prior knowledge will be incorporated into the recursive autoencoder which can generate the document vectors. Finally, the softmax function in the classification model will compute the probability of each possible keyword given each document.

The main difference between our model and original semi-supervised recursive autoencoder includes two parts. First, the input the autoencoder is changed to the vector of words generated by word2vec instead of randomly parameters. Second, we also replace the tree structure which is generated by the greedy algorithm with the strucure of the grammar tree.

Besides, we also propose the co-occurence model. In the model without co-occurrence model, we only consider top 200 frequent keywords. But in the co-occurrence model, we take into account all the keywords. The reason for this is we want to overcome the limitation of supervised learning. With supervised learning, we will not be able to predict any keyword from outside of top 200 keywords. With co-occurrence, we predict the keyword from outside the top 200 tags as well. We first compute the probability of a keyword in the top two hundred keywords, taking the document as the prior $P(L_t|doc_i)$. Next we compute the conditional probability of all the keywords given that we have observed a keyword co-occurring with a top 200 keyword $P(L_o|L_t)$. Using these two relations, we can obtain the equation for the probability of any keyword occurring in a document. For the details about the co-occurence model, please refer to the experiment part.

In the Figure 6, the red rectangle means the model we generate. The yellow rectangle means the input and output of our model.

$$P(L_o|doc_i) = \sum_{i=1}^{T} P(L_o|L_t).(L_t|doc_i) \tag{4}$$

,where L_o : Generic keyword(includes all keywords, those not in top 200 as well), L_t : Keyword in top 200 keyword, doc_i : Document i in Test Data

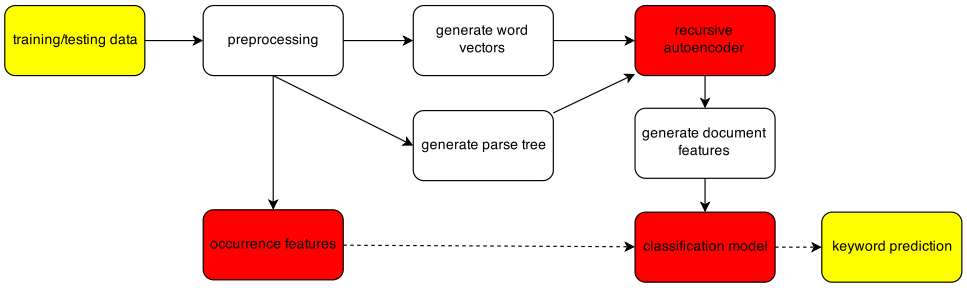

Figure 6: Our frame for keyword extraction. Here we have done some work on the yellow part: Recursive Autoencoder.

5 Experiment and Results

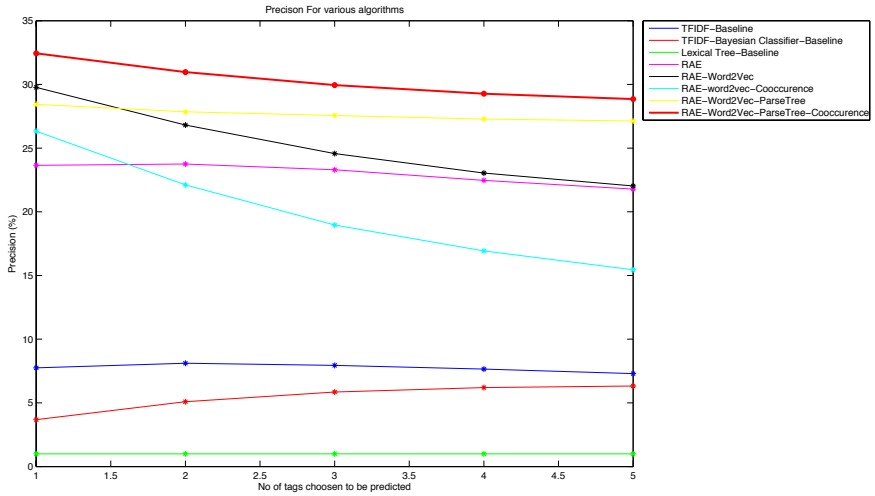

Figure 7: The precision comparison. Here we see our model performances best and when the number of keywords increase, the difference between the original RAE model and the one using word2vec is less and less.

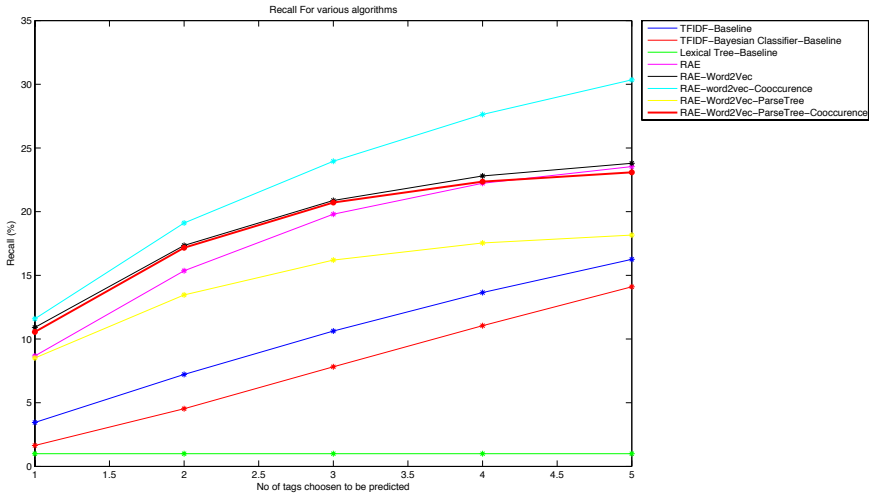

Figure 8: The recall comparison. Here similar conclusion can be drawn as that of the precision comparison.

5.1 Dataset

We had identified data from stackexchange [1] to be a suitable data set for our project.

Of the 20 GB, available to us, for the feasibility of the experiment, we are using only 95 MB of data, which we have cleaned and formatted for our applications, on about 5 topics for our evaluation. Although the data set seems to be small compared with original one, we felt this was enough for testing purposes. Later we will try to **increase the data set** when our model is finished.

Here is an interesting question, why we choose several topic and not only one topic as small data set to test our model? In fact we did that, but found that it's not a good way to do testing of the task like keyword extraction. For a single topic, some words will occur frequently, for example, in the topic of *apple*, the words like *osx* and *iphone* will occur frequently in most documents which according to *tf-idf*, they are not important words. But in fact they are the keywords for many documents in this topic. And if we train the model on only a single dataset, then if the model only predict the top frequently occurring keywords, it can still gain good performance. That's the reason we use several topics as dataset here.

The last step is to divided the data set into training dataset and testing dataset. We random sample 20% as the testing dataset and the rest are the training dataset. The number of documents in each data set is: 37935 for testing data set and 152537 for training data set.

5.2 Baseline Algorithm Results

To evaluate the performance of our approach compared to existing techniques, we have used Precision and Recall rates as our benchmarking standards. We have formally defined these parameters as

$$\begin{aligned} Precision &= \frac{sizeof(A \bigcap B)}{sizeof(A)} \\ Recall &= \frac{sizeof(A \bigcap B)}{sizeof(A)} \end{aligned} \tag{5}$$

where A is the set of predicted tags and B is the set of actual tags.

We have implemented TF-IDF and Bayesian Classifier approach as described by the survey for Keyword Extraction on our data set. With plain vanilla TF-IDF, the average precision and recall rates were less than 10 %. However, after preprocessing the data by using a stemmer, the accuracy is a modest 15-20% on topics in our data set for as the best performer.

We have used portland stemmer, available as part of open source NLTK(Natural Language Processing Toolkit) project.

The Bayesian Classifier is similar to TF-IDF approach except that it takes into account the position of the word in text. The weight of each term is reduced as the log of the its position from the beginning of the sentence.Also since the documents we used are not so large documents, the depth of the word into the document did not affect the performance drastically compared to TF-IDF. It's results were similar to TF-IDF.

As part of survey paper[3], we found out that the existing standard for key word extraction uses Lexical Tree approach and hence we have summarized its performance on our data set as well. As per the lexical tree approach, the precision and recall were very poor, less than 1 % when tested against our data set. The reason for this can be attributed to the fact that our documents are very small posts with about a paragraph in them. Siber and McCoy[5] had implemented this feature for larger documents such as conference papers , essays etc and they would have probably had a document with more than 25 nouns, contrary to this, we only had 4-8 nouns per document on an average, hence we could not build a bigger graph for our documents.

5.3 Analysis of Our Model

Our work is based on the open source software [2]. For the midterm report, we use the tool to do some simple checking and we also modify some parts of the it to make use of the word vectors generated by *word2vec*[3], another tool developed by Google.

[1] `https://archive.org/download/stackexchange/stackexchange_archive.torrent`

[2] https://github.com/sancha/jrae

[3] https://code.google.com/p/word2vec/

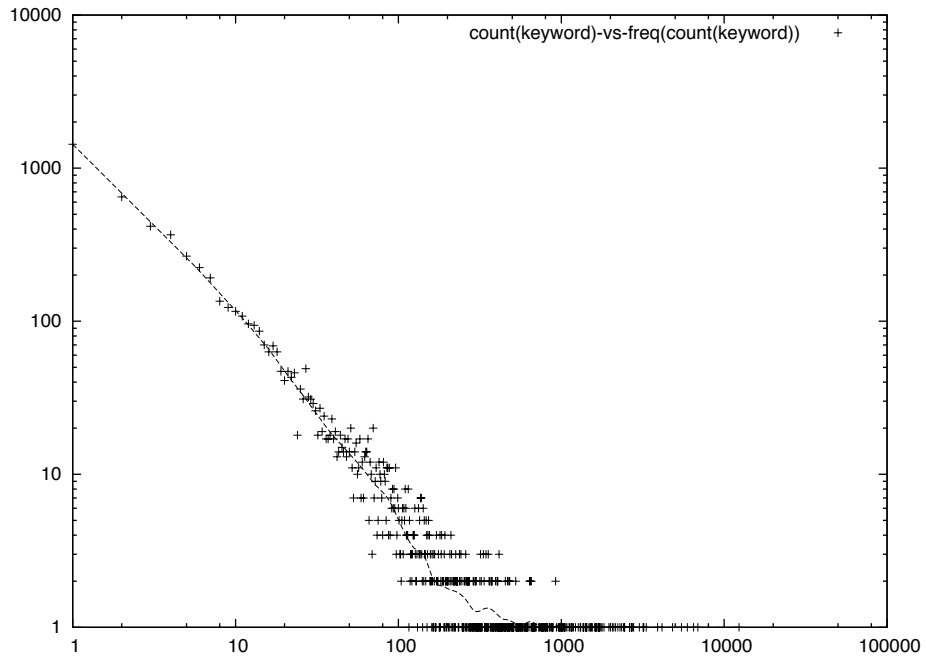

Figure 9: The distribution of keywords follows Zipf's law. Here the x-axis is the count of the keyword and the y-axis is the frequency of the count of keywords. Drawn in log-log scale.

Before discussing about the experiment, we will introduce the method about training data preprocessing. We analyze the distribution of the keywords in the dataset, and we find it follow Zipf's law from Figure 9. So we can't train the model with all dataset. And here, our model is a supervised model, and it's a multi-class classification task so it's better not to train the model on all the keywords. Here, we just choose the top 200 frequently keywords and for each keyword we sample 100 documents.

Besides, our task is a little different with the classical classification task. For each document, we may give more than one keyword, because it's common for a document having more than one keyword. So when predicating, we generate the belief of each keyword for each document and choose the top-n keywords.

We do the testing on these settings:(i) RAE (ii) RAE-word2vec (iii) RAE-word2vec-cooccurence (iv) RAE-word2vec-ParseTree (v) RAE-word2vec-ParseTree-cooccurence(final model) . Here RAE means Recursive Autoencoder; word2vec means using the pre-trained parameters for word vectors in recursive neural network; ParseTree means, when training the model with RNN, the tree structure is fixed by a parse-tree model; co-occurrence here means when doing classification, the co-occurence features calculated on all the training data will be used to spread the result of the top 200 tags to all the tags.

From the Figure 7 we see that our model performs better than all the other models. The testing is done on the testing dataset mentioned above and because we want to simulate the true keyword extracting task in real word, although some models are trained on only 200 tags, we still testing them on the data that may contain the keywords not in the sampled training dataset. Besides some more advanced methods such as using the cooccurence feature, trained on all the training dataset, to predict the possibility of other tags will also predict the keywords not in the sampled training dataset. So no filtering of keywords in test dataset will also makes them comparable.

The following is analysis about the precision curve.

- Baseline methods are worse than any of our models.
- RAE-word2vec and RAE seems to have the same asymptotic results. We don't know the true reason behind

this phenomenon, but here is some guess about it. Since these two kinds of model only predict the tags in the sampled training data (top 200 highest frequency tags) and we also do filtering when doing the prediction, when the model is asked to give 5 tags, for a lot of documents, the model can't give them five tags in fact. So they eventually converge to a constant. Besides when using word2vec to pretrain the word vectors of the model, only 10-dimensional features are trained, because the training process of RAE is slow. 10-dimensional word vectors make the effect of word vectors not that important. We could not generate higher dimensional word vectors as the time required to generate them was too high.

- After fixing the parsing tree, both models(RAE-word2vec and RAE-word2vec-cooccurence) behave better. The model used to predict the parsing tree is trained on other corpus, which contain more information. Also fixing the parsing tree makes the model less complex, because in RAE-word2vec, the parsing tree will be created in the model itself. The training data we use is small, and less complex model do favor to our task here.

- A note here is that, our final model is not trained on the stemmed data which may make the result not comparable. The reason that we don't train the model on the stemmed data is that because we use the parsing tree and the model (trained by the author) predicting the parse tree is not trained on the stemmed data. Although to some degree the results are not comparable, we believe if trained on the stemmed data, our model will behave better. There are some words/tags in the dataset when stemmed, they will become the same which makes the task easier. For example such as "iphone-4" and "iphone4s" and for both tags, if stemmed, they will become "iphon". So our final model has some complex features(parse tree.. etc) are incorporated into it while training, and tested using the same. That's why we believe it should behave better.

The following is analysis about the recall curve.

- There is some similar phenomenons as that of the precision curve. One is that our methods beat all the baseline. The other one is the curve of RAE-word2ve-co-occurrene and the reason why it beats the RAE-word2vec is stated above. And the reason why it performs better than our final model we think it's because of stemming which make the task easier for RAE-word2vec-co-occurence.

- The reason why the RAE-word2vec-ParseTree is worse than not using parse tree is also because of stemming.

- It deserves noticing that RAE, RAE-word2vec and the final model all have similar curve for recall. And since the training data set for RAE and RAE-word2vec is stemmed, their precision is also similar. But our final model with co-occurence feature, of which the training data is not stemmed, performs the best.

6 Conclusion

We would like to conclude that deep learning techniques certainly improve keyword extraction by a significant factor. We also witnessed that adding prior knowledge from the parse tree to the semi-supervised recursive auto-encoder, improves it even more.

We have studied and implemented some baseline algorithms. We also studied the recursive autoencoder algorithm mentioned in the proposal. What's more we have done some experiments based on recursive autoencoder and also integrated *word2vec* with recursive autoencoder.We have added the parse tree to the model and tested it separately. To overcome the limit of supervised learning, the co-occurrence of the keywords were used to improve the recall and precision. We could not,however implement some goals such as increasing the dimension of word vectors and implementing hierarchical clustering , due to computing constraints and complexity involved in the tasks.

Besides, because of the problem of stemming, it makes the results not comparable. We didn't notice it until doing the evaluation of RAE-word2vec-ParseTree. Therefore, it is difficult to do the previous experiment again. But the final model outperforms the models trained on the stemmed data. In addition, we also notice that doing stemming can make the task simpler. For instances, Without stemming, the result of TF-IDF is very bad.

7 Stretch Goals

We were limited in time and resources in this project. However we would like to propose some stretch goals for our project.

1. Increase the dimension of word vectors and training dataset size and do testing. This will certainly give us more accurate features from our data, and we expect the performance to improve. We would like to try this in a cluster environment.

2. Using SVR or hierarchical model instead of softmax in the classification part. Due to paucity of time, we could not implement it in this project.

8 Appendix

- Yi Cheng: Responsible with modifying RAE and integrate it with word2vec and parsing tree; doing the experiment with the models proposed.
- Anoop Hallur: Being familiar with the baseline algorithms and implementing them; implementing the co-occurrence features and doing evaluation.
- Xiaoqiu Huang: Focusing on the theoretical part, covering the details of RAE, Parsing Tree, Word2vec; working with Anoop together to implement the lexical tree baseline.

References

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[2] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[3] Brian Lott. Survey of keyword extraction techniques. *UNM Education*, 2012.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[5] H Gregory Silber and Kathleen F McCoy. Efficiently computed lexical chains as an intermediate representation for automatic text summarization. *Computational Linguistics*, 28(4):487–496, 2002.

[6] Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with compositional vector grammars. In *In Proceedings of the ACL conference*. Citeseer, 2013.

[7] Richard Socher, Yoshua Bengio, and Christopher D Manning. Deep learning for nlp (without magic). In *Tutorial Abstracts of ACL 2012*, page 55. Association for Computational Linguistics, 2012.

[8] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 129–136, 2011.

[9] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 151–161. Association for Computational Linguistics, 2011.