Information Retrieval and Text Classification on Mandarin Chinese News Report

Sheng-Syun Shen[1], Wei-Tse Lee[2], and Ching-Ta Wu[3]

[1]Graduate Institute of Communication Engineering
[2]Graduate Institute of Networking and Multimedia
[3]Graduate Institute of Electronics Engineering

2016.01.08

In this project, we proposed some approaches for information retrieval and text classification, which are two sub-tasks in Natural Language Processing (NLP).

1. INTRODUCTION

In this class, Information Retrieval and Extraction, every three students are grouped up to implement term project. This term project is divided into two parts, Phase 1 and Phase 2. Phase 1 is due at mid-term, and Phase 2 is due at the end of semester. The project goal is to encourage students working on a research issue, reading relevant papers, implementing algorithms, making flexible use of information retrieval (IR) and information extraction (IE) skills learned in class, and brainstorming, etc. Phase 1 is for query by news stories. Students are given some news stories, and they have to find relevant news according to these given news stories. After the mid-term exam, each group is asked to prepare a brief presentation to share ideas with other groups. Phase 2 is for news-stories agencies classification. Groups are added to Kaggle, which is an on-line platform for data-prediction competitions to implement Phase 2. Students are given a lot of news stories with agency-class labels as training data, and they have to identify the test news stories' class labels, and the events in the test news stories are not in training data. The performance of Phase 1 and Phase 2 are evaluated by mean-value precision (MAP) and accuracy, respectively.

2. PHASE I

In this section, we would like to focus on the information retrieval task. We proposed a neural-based word embedding model, and also compared with two existing classical models, including bag-of-word model and language model. In 2.1., we will introduce three models we applied in this task, and in 2.2., we are going to evaluate the performance of each model on our dataset - Chinese news report. Then, we will discuss the experimental results in 2.3..

2.1. METHODOLOGY

2.1.1. System Flow Chart

We are going to introduce how our models work in the information retrieval task.The system flow chart is shown in Figure 1. Given a bunch of documents and queries, first we will conduct some text pre-processing procedures, especially for Mandarin Chinese corpus. After that, both queries and documents should be mapped to the same spaces using the models we proposed so that we can conduct

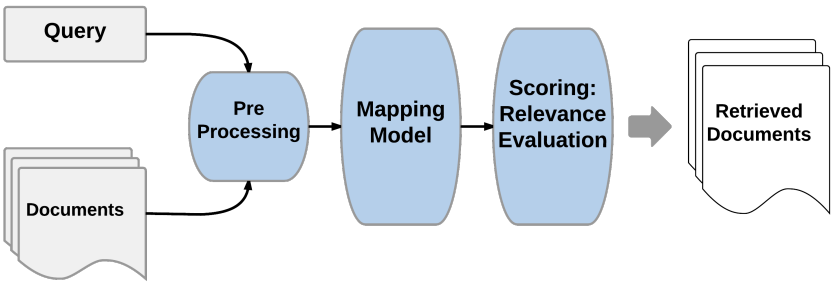

Figure 1: System flow chart for our proposed retrieval models.

content similarity measurements more easily. However, the evaluation criterion may vary from different mapping models we use. Finally, we can get a list of retrieved documents, which is sorted by the relevance with queries. In the following section 2.1.2., we will introduce the pre-processing method we used, and from section 2.1.3. to 2.1.5. show the mapping models and the corresponding relevance evaluation method.

2.1.2. Pre-processing

The most important pre-processing task for Mandarin Chinese corpus is text segmentation. Text segmentation is the process of dividing written text into meaningful units, such as words, sentences, or topics. The problem is non-trivial, because for some written languages (e.g., English and French), sentences are written with explicit word boundary markers, such as the word spaces. Without these word spaces, the meaning of sentences might be ambiguous. For example, there's a saying that goes "有兩種人最容易被甩，一種是不知道什麼叫做愛，另一種是不知道什麼叫做愛。". We can understand the sentence "不知道什麼叫做愛" in two ways, "不知道、什麼、叫做、愛" and "不知道、什麼、叫、做愛", respectively, which causes two different meanings "Don't know what love is" and "Don't know how to have sex".

We choose to use a third-party library "Jieba" to do the trick due to its powerful capability when handling Chinese vocabularies. Note that we use a different dictionary instead of the default one since it only performs well on Simplified Chinese but not on Traditional Chinese. We also remove the punctuation to avoid the meaningless vocabularies.

2.1.3. Bag-of-word Model

The bag-of-word model is a simplifying representation used in natural language processing and information retrieval. In this model, a text (e.g., a sentence or a document) is represented as the bag (multi-set) of its words, disregarding grammar and even word order but keeping multiplicity. The bag-of-words model is commonly used in methods of document classification, where the frequency of occurrence of each word is used as a feature for training a classifier. After pre-processing (text segmentation) on these given news stories along with queries using "Jieba", we turned each of them into a set of words without punctuation. Next, we collected these words in each set to generate a vocabulary corpus. Now that we have a vocabulary corpus, we can convert these news stories and queries into vector space where each term of the vector is indexed as our index vocabulary. We use the term-frequency to represent each term in our vector space. For example, consider the following two sentences:

```
Sentence 1:  "The cat sat on the hat"
Sentence 2:  "The dog ate the cat and the hat"
```

From these two sentences, our vocabulary is: {`the, cat, sat, on, hat, dog, ate, and`}. To get our bag-of-words, we count the number of times each word occurs in each sentence. In

Sentence 1, "the" appears twice, and "cat", "sat", "on", and "hat" each appear once. so the feature vector of Sentence 1 and 2 are:

```
Sentence 1:  { 2, 1, 1, 1, 1, 0, 0, 0 }
Sentence 2:  { 3, 1, 0, 0, 1, 1, 1, 1 }
```

Then we use the definition of inverse-document frequency (idf):

$$idf(t) = log\frac{|D|}{1 + |d : t \in d|}, \tag{1}$$

where $|d : t \in d|$ is the number of documents where the term appears, when the term-frequency function satisfies $tf(t, d) \neq 0$, we' re only adding 1 into the formula to avoid zero-division.

The formula for the tf-idf is then equals to:

$$tfidf(t) = tf(t, d) \times idf(t), \tag{2}$$

and this formula has an important consequence: a high weight of the tf-idf calculation is reached when you have a high term frequency in the given document (local parameter) and a low document frequency of the term in the whole collection (global parameter).

Finally, we use cosine-similarity (dot product) to calculate the similarity between news stories and queries. Cosine Similarity will generate a metric that says how related are queries and news stories by looking at the angle instead of magnitude. Now that we have a Vector Space Model of news stories and queries modeled as vectors (with TF-IDF) and also have a formula to calculate the similarity between different news stories or queries in this space. We use Python module —scikit-learn to do these things mentioned above in practice.

2.1.4. Language Model

Based on paper (Javanmardi et al., 2010), we choose to use bigram language models to solve the problem. Firstly, we need to split all documents to combination of vocabularies.

After vocabulary segmentation, the next step is to generate bigram language models for each document. We calculate not only frequency of bigram terms but also unigram terms since we need to use unigram terms as bases in our score formula. Having these models, we then examine each query, which also pass the vocabulary segmentation process, word by word and calculate relevant score for each document. The score formula we used here is reference to paper (Javanmardi et al., 2010) which is as following:

$$\begin{aligned} P(q_i|q_{i-1}, D) = (1 - \lambda_1)[(1 - \lambda_2)\frac{f_{qi,D} + \mu_1/|V|}{|D| + \mu_1} + \lambda_2\frac{f_{q_{i-1}q_i,D} + \mu_2/|V|^2}{f_{q_{i-1},D} + \mu_2}] \\ + \lambda_1[(1 - \lambda_3)\frac{f_{qi,C} + \mu_3/|V|}{|C| + \mu_3} + \lambda_3\frac{f_{q_{i-1}q_i,C} + \mu_4/|V|^2}{f_{q_{i-1},C} + \mu_4}] \end{aligned} \tag{3}$$

where q_i represents the i-th word in query, D means the target document, λ and μ are corresponding parameters, $|V|$ is the dictionary size and f means the unigram or bigram frequency.
The parameters we use are as same as the ones in paper, which is (λ_1, λ_2, λ_3, μ_1, μ_2, μ_3, μ_4) = (0.24, 0.29, 0.94, 1800, 400, 792, 900). In order to get the score for specific document, we take logarithm for each term-term probability and summarize it. Therefore, the document with a larger score will be assumed as more relevant.

After examination, we choose top 10 relevant documents to perform as feedback for each query. Combining these 10 documents as a new query and redo the scoring process. Finally, we choose top 100 relevant documents as our final result.

2.1.5. Word-embedding Model: Paragraph Vector

Applying neural-based word embedding system to Natural Language Processing tasks is a new trend nowadays. Since Tomas Mikolov's `word2vec` has made public to the world (Mikolov et al., 2013), it's been widely adapted and the results proved that word embedding models can make progress on many domains, e.g., text summarization, question answering, semantic analysis, and so on. In our task, documents and queries should be mapped to the same vector space so that we could measure the relevance score. Word-embedding models can reach the goal, and surprisingly they can also preserve the semantic relationships between documents. We adapted the advanced application of `word2vec`, which called `sentence2vec`.

`sentence2vec` is an implementation of Quoc Le's work (Le and Mikolov, 2014), which handles not only word embedding but also sentence embedding. They add one more input vector to the skip-gram model (Mikolov et al., 2013), and this vector is the one-hot encoding of documents, which means the index of vector can represent the ID of a document. Both input vectors would project to the same hidden layer, so they can train jointly. The model structure is shown in Figure 2. We set the number of dimensions to 300, and the length of context-window to 8. The corpus we used to train the word-embedding models is previously collected from PTT, the biggest BBS station in Taiwan, and it has 71,979,326 words in total. With a new scheme of vector representation for every document and query, we then use cosine-similarity to measure the relevance score, which has already been described in section 2.1.3..

2.2. Evaluation

2.2.1. Dataset Description

We conduct our retrieval model on a Mandarin Chinese News Dataset, which is collected from Taiwan daily news and written in traditional Chinese. This dataset contains 163132 news reports, and each news has 304.93 words in average after the pre-processing of word segmentation. Besides, we have 17 queries, and each one has the average of 1816.18 words.

2.2.2. Evaluation Methodology

We report 100 relevant documents for each queries for evaluation. We select mean average precision (MAP) as our evaluation methodology, which is also called "average precision at seen relevant documents." MAP will determine precision at each point when a new relevant document gets retrieved, and set the precision to 0 while no relevant document was retrieved. MAP determine average for each query, then average over queries as equation 4:

$$MAP = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{Q_j}\sum_{i=1}^{Q_j}P(doc_i), \tag{4}$$

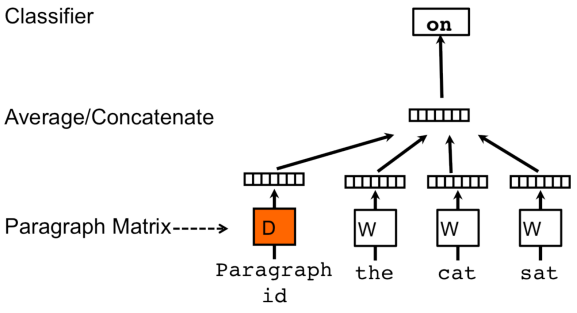

Figure 2: Paragraph vector model structure.

Table 1: The experiment results for news story retrieval.

Retrieval Model	**MAP**
Language Model	0.7502
LM + Embedding	0.7600

where Q_j is the number of relevant documents for query j, N is the number of queries, and $P(doc_i)$ is the precision at ith relevant document.

2.2.3. Experimental Results

We only report two results because the evaluation scores were given by the course TA. The results is shown in Table 1. The first row shows the language model result, which is the top-5 result in class. The second row is the combination of language model and word-embedding model results. In Vulic et al.'s paper(Vulić and Moens, 2015), we found that embedding models can't work better than traditional retrieval models by themselves. So we report the results by combining both models rather than using unique model.

2.3. DISCUSSION

In Table 1, we can found that considering bi-gram language model may perform better than using only uni-gram language model (comparing to the other results in class), which shows the word order and context information are important for understanding a document. Furthermore, combing language model and word-embedding model can also make progress. We may conclude that while preserving the semantic and syntactic information in vector space formulation for documents and queries, the performance of retrieval results would be better.

3. PHASE II

In this section, we are going to solve the text classification problem. Given a bunch of news stories and their corresponding newspaper office, we are going to predict the newspaper office name while we only know the news story content. We adapt both traditional machine learning and deep learning methods to solve the classification problem. In section 3.1., we will show the methodology we proposed, and section 3.2. reports the classification result on testing set. We will also discuss the results in section 3.3..

3.1. METHODOLOGY

3.1.1. Learning by Tree

We have tried to use tree-based learning model to achieve the goal in phase II. The first thing we need to do is to extract the features in training data. Since the training data is plain text, we use the same vocabulary segmentation process we did in Phase I to get the segmented documents. Having these documents, this time we use third party library "Scikit-learn" to get the tf-idf matrix from vectorizer it provides by feeding it the segmented documents. We use the same vectorizer to extract the features from segmented documents which generated from testing data. Thus, we get features for both training data and testing data with same dimensions.

Note that we have generated several different dimensions of features by controlling the threshold of minimum document frequency in order to verify if it will make influence to the result. Besides reduce the dimension by frequency, we also test dimension reduction techniques such as random projection. The detail will be discussed in evaluation session.

For learning models, we have tested gradient boosted regression tree and random forest. We use "Scikit-learn" library to implement these two classifiers.

3.1.2. Learning by Deep

After surveying lots of paper, we found that most text classification tasks can be solved by Support Vector Machine (SVM) classifier(Joachims, 1998)(Pilászy, 2005). This can be a powerful baseline. However, the computational cost of SVM is huge, while facing large-scale text classification problems, SVM might not be the best solution nowadays.

Therefore, we imported deep neural network (DNN), which is the so-called "Deep Learning"(Arel et al., 2010), trying to solve this classification problem. Deep neural networks can handle higher model complexity, so it is easier to fit the seen training data and therefore it can predict the unseen testing data more accurately. Besides, neural networks are mostly solved by stochastic gradient descent, and can be modeled by GPU acceleration. As a result, the computation cost of DNN is much smaller than SVM. We implemented our customize neural network by `Keras`, a neural network library which is written in python and be capable of running on `Theano`. The detail of model information will be described in the next section.

Solving classification problem can not just by using powerful machine learning tool, having a better vector representation method to describe our documents is also important. We proposed three kinds of feature representation schemes, which is shown as follows:

- **Bag-of-word**: This representation scheme has already been described previously. The number of dimensions is 719,273, and each element in feature vectors is calculated by tf-idf value. Due to the high dimensionality and sparsity of feature vector, we also set a minimum document frequency threshold to filter out some rarely-used words. Thus, we have a new feature vector set, and the number of dimensions reduced to 129,122.

- **Latent Dirichlet Allocation**: LDA (Blei et al., 2003) is a generative model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's creation is attributable to one of the document's topics. We used `GibbsLDA++`(Phan and Nguyen, 2007), a C/C++ implementation of LDA to extract topical features. It is very fast and is designed to analyze hidden/latent topic structures of large-scale datasets including large collections of text documents. We set the number of topics to 100, and training iterations to 100 as well.

- **Sentence-embedding**: `sentence2vec` has been introduced in section 2.1.5., and we use the same settings as well. Note that we also have tried to implement neural-based document embedding (Huang et al., 2014) and topical embedding (Liu et al., 2015) models. However, the performance isn't that ideal so we discarded them, and probably we will try to revise them in the future.

3.2. EVALUATION

3.2.1. Dataset Description

We use the same dataset in the previous task, a Mandarin Chinese News Story Set. All the news stories are released by three Newspaper agencies: LTN(自由時報), CTS(中國時報), and APD(蘋果日報). The number of training data is 138,005 and 1,805 for testing data.

3.2.2. Tree-based Results

For tree-based learning models, gradient boosted regression tree perform inefficiently and the result it produce is not so good. The reason might be the unsuitable parameters. Since the training time is too long, we have not done much effort to get the best parameters for it. On the other hand, random forest have a remarkable performance and the result is much better. Table 2 shows our experiment result.

Table 2: The experiment results for tree-based learning models. Do the experiement on hardware with Intel Xeon E5-2650 v2, 2.6GHz CPU, 128G RAM and Nvidia Tesla K20m * 2

Learning Model	**Dimension Reduction**	**Remark**	**Time**	**Accuracy**
GBRT	-	-	about 12 hrs	0.452
Random Forest	-	10 estimators	about 1 hrs	0.649
Random Forest	Random Projection	-	about 0.5 hrs	0.595
Random Forest	-	500 estimators	about 3 hrs	0.761
Random Forest	-	3000 estimators	about 10 hrs	0.764
Random Forest	-	500 estimators with L1 norm	about 3 hrs	0.769

Table 3: The experiment results for NN-based learning models. Do the experiement on hardware with Intel Core i7-4790, 3.60GHz CPU, 20G RAM and Nvidia GeForce GTX 960

Feature Schemes	**DNN Struecture**	**Epoch**	**Time**	**Accuracy**
(a) BOW	128×1024×1024	05	about 1 hr	0.818
(b) BOW	128×1024×1024	30	about 6 hrs	0.839
(c) BOW Reduced	128×512×512	35	about 30 mins	0.838
(d) BOW Reduced	256×1024×1024	10	about 15 mins	0.804
(e) BOW Reduced	256×1024×1024	15	about 20 mins	0.806
(f) BOW Reduced	64×128×256	05	about 3 mins	0.827
(g) BOW Reduced	64×128×256	10	about 6 mins	0.827
(h) Sentence2vec	2048×1024×512	10	about 3 mins	0.697
(i) LDA	128×256×512	05	about 7 mins	0.673
(j) 2 BOW	-	-	-	**0.846**
(k) BOW+LDA	-	-	-	0.845

3.2.3. NN-based Results

Table 3 shows the experimental results using neural networks to perform training procedure. The first column introduces different form of feature vectors. The second column represents the structure of hidden layers. We have 3 layers for each experimental set, and the numbers means the dimensions of hidden layers. The third column shows the number of training iteration. Rows (a) and (b) used the full bag-of-word features while rows (c) to (g) chose the reduction version. We can see that Row (h) is for sentence-embedding and row (i) is for LDA. Notice that row (j) is the combination of rows (b) and (c), and row (k) is from rows (b) and (i). We used equal-weight interpolation to perform the combination procedure. All the neural networks are trained with stochastic gradient descent optimizer, setting learning rate as 0.1, momentum as 0.9, and using `relu` as activation function.

3.3. Discussion

For the tree-based learning models, we found that in dimension reduction part, using random projection will decrease the accuracy but will increase the efficiency. But by controlling minimum document frequency, we can increase both accuracy and efficiency, so it will be a better alternative to do if the original dimension is too large to process. Observing the table, we can also find that there is little improvement if we change number of estimators from 500 to 3000. In conclusion, sufficient number of estimators have a lot to do with the accuracy. And if we control the minimum document frequency to a proper number, we can have a better performance and quality.

While for the nn-based models, we compared the results of different training model complexity. We found that not always the higher complexity would cause higher accuracy. For example, rows (d) and (g) used the same feature vectors but different model structures. When they were trained in the same epochs, the higher complexity caused lower accuracy. We may conclude that the real complexity for this classification problem isn't that high, so too deep or too wide neural networks would lead to

over-fitting. Besides, the more epochs your trained doesn't promise the better performance you can get (rows (f) v.s. (g)). Sentence-embedding and LDA features didn't perform ideally, which might due to the characteristics of text classification. When someone classifies documents into categories, he/she will examine the keywords for every document. That's why bag-of-word features can beat the others. However, we can found that combining bag-of-word and LDA features can make some progress, and ensemble the top-2 bag-of-word results will achieve highers score.

4. CONCLUSION

We examined information retrieval and text classification problems in this project, which are two subtasks in Natural Language Processing (NLP). We have tried different learning models, feature extraction schemes, and some parameter tuning methods. We found that although some feature extraction methods are really basic, they can still be very powerful, such as language model and bag-of-word. However, if we consider the semantic and syntactic information by leveraging other types of models, and combine with the traditional method, we can then achieve big progress.

REFERENCES

AREL, I., ROSE, D. C., AND KARNOWSKI, T. P. 2010. Deep machine learning-a new frontier in artificial intelligence research [research frontier]. *Computational Intelligence Magazine, IEEE 5,* 4, 13–18.

BLEI, D. M., NG, A. Y., AND JORDAN, M. I. 2003. Latent dirichlet allocation. *the Journal of machine Learning research 3*, 993–1022.

HUANG, C., QIU, X., AND HUANG, X. 2014. Text classification with document embeddings. In *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*. Springer, 131–140.

JAVANMARDI, S., GAO, J., AND WANG, K. 2010. Optimizing two stage bigram language models for ir. In *Proceedings of the 19th international conference on World wide web*. ACM, 1125–1126.

JOACHIMS, T. 1998. *Text categorization with support vector machines: Learning with many relevant features*. Springer.

LE, Q. V. AND MIKOLOV, T. 2014. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*.

LIU, Y., LIU, Z., CHUA, T.-S., AND SUN, M. 2015. Topical word embeddings.

MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. S., AND DEAN, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

PHAN, X.-H. AND NGUYEN, C.-T. 2007. Gibbslda++: Ac/c++ implementation of latent dirichlet allocation (lda).

PILÁSZY, I. 2005. Text categorization and support vector machines. In *The proceedings of the 6th international symposium of Hungarian researchers on computational intelligence*. Citeseer.

VULIĆ, I. AND MOENS, M.-F. 2015. Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 363–372.