MDM-2021 Project Report

Ming Jiang (1016239), Gengcong Yan (1009903)

December 23, 2021

1 Data Preprocessing

Data preprocessing is an essential step in text mining, improving text format in data to remove unnecessary errors and noise. The application of preprocessed data can enhance the performance of the model. In our project, data preprocessing includes a combination of text, tokenization, stemming and removing meaningless words, introduced as follow.

1. **Combination of text**. Firstly, the text part of data consists of *title* and *abstract.* we combined the contents in *title* and *abstract* together to represent the current paper.

2. **Tokenization**. Tokenization is a method of separating the long text into smaller units of tokens. Usually, one word represents one token. It's a common step in NLP for ease of representation and computation later.

3. **Stemming**. Stemming is the process of reducing the word to its original stem. Different forms of variation in words can exist in different expressions. But They only represent one meaning in the actual analysis. So stemming will reduce them to their most initial root state, representing the same meaning. After the process, it reduces the number of cases where the same meaning but different forms of words are used, helping computation of the analysis later. We perform the Porter Stemming Algorithm[1] in the NLTK package for text.

4. **Removing unnecessary words**. Unnecessary words here refer to stopwords, punctuation and numbers. Stopwords are frequently occurring words in the text but with little information about the content. They are not necessary for understanding the text content. Punctuation without information should also be deleted in the text. At the same time, numbers in text present little information. Thus we remove them as well. We perform English stopwords of the NLTK package in preprocessing.

2 Feature Extraction

In this section, we use the basic tf-idf model for document vectors learning. Except for this, we also do a further attempt to use the **word2vec**[2] to represent the document. This part is shown in the Appendix A.

Table 1: NMI Value Comparison of Different Clustering Methods with tf-tdf Vectors

	dim=64	dim=128	dim=256	dim=512
kmeans-L2	0.6947	0.6705	0.7105	0.7414
spt-RBF	0.7816	0.7726	0.7608	0.6697
kmeans-cosine	0.8022	0.8095	0.8104	**0.8156**

2.1 TF-IDF Model

We use the predefined class *TfidfVectorizer()* in *sklearn*, and we set the parameter *smooth_idf* as *True*, and meanwhile use the *L2-norm* to normalize the tf-idf vectors. After checking the source code of *TfidfVectorizer()*, we figure out that the smoothed tf-idf transformation equation is shown as following:

$$tf - idf(t, d) = tf(t, d) * log[\frac{(1 + n)}{(1 + df(d, t)}] + 1 \tag{1}$$

where n is the total number of documents and $df(t)$ is the document frequency of word t. $tf(t, d)$ is calculated as the number of documents containing word t.

As a results, we finally get a multidimensional tf-idf feature matrix $D = data\_size \times vocab\_size$, which in our dataset, $data\_size = 1332$, $vocab\_size = 10260$. Each feature for document d represents the tf-idf score for word t. The dimension equals the whole vocabulary size of all words in all documents.

2.2 Dimension Reduction

Since most entries of the tdf-vectors are zero, D is a very **sparse** matrix. In addition, the high dimension makes most distance metrics not so efficient when doing the clustering and classification. As a result, here we use LSA to reduce the dimension of D. We made experiments for different data dimensions to obtain the most proper ones.

3 Evaluation

3.1 NMI Value Comparison

In this section, we choose different clustering methods to check the results. We use k-means with Euclidean distance as the baseline and compare the results with spectral clustering methods with RBF kernel method, as well as k-means with cosine similarity matrix. We simulate the grid-search to search for the best hyper-parameters in our models. As for the spectral clustering method, we tested different *gamma* for the RBF kernel and chose $gamma = 0.8$ as the best display in the Table. We ran **10** times of each method and recorded the average NMI scores. The final mean evaluation results are shown in Table. 1.

The results show that spectral clustering is better than kmeans with the Euclidean distance metric. However it became sub-optimal when the dimension increased. K-means with cosine similarity metric outperforms all other methods because cosine similarity is better to measure the text features.

3.2 Content Analysis

Word Cloud is utilized in the project for finding relevant keywords[3]. Word Cloud is a visual representation of different words in a graph, and the more critical words will occupy a more prominent position, proving the importance of words in the text. So The words we focus on firstly in Word Cloud can be considered as keywords in the text under analysis. There are also other mathematical solutions detecting keywords like counting frequency. But the changes in numbers are more abstract and difficult to understand. On the other hand, the word cloud makes it intuitive to understand in a short time by comparing words directly on the canvas and visually displaying keywords in the text.

We choose our best clustering model–**K-means Cosine** and cluster the text data into five groups. The text data combine a corresponding title and abstract in papers. We fed the Word Cloud model with five groups of text data, generated five graphs with relevant keywords in various subjects in Appendix A below. Some words are missing the letter "e" at the end, but this does not affect our understanding. This happens because stemming will convert all words to their original root state. The keywords and subjects in clusters can be arranged as follow:

1. **Computer Vision** : image, computer vision, dataset, train, visual, image detection.
2. **Robot Design & Simulation**: robot, control, design, sensor, learn, simulation.
3. **Compiler Design**: compile, language, optimize, implement, code, parallel ,system analysis.
4. **Database**: query, SQL, database, system design, use.
5. **Information Security & Cryptography** : security, encryption, quantum, scheme, protocol, attack ,network, cryptography, safety.

The word cloud representations of above clusters show in Appendix B.

4 Conclusion

In the project, after preprocessing the text data, we use TF-IDF scores of words to represent the document features. Next we chose different clustering methods to do to the document clustering task with NMI score as the metric. We test K-means with euclidean distance and cosine similarity matrix, spectral clustering with RBF kernel in the project. In our experiments, K-means with cosine similarity outperforms all other methods as $NMI = 0.8156$ because cosine similarity is better for measuring text features. Lastly, the generation of word cloud in 5 clusters extracts the keywords in different groups for better understanding specific topics.

5 Execution instructions

Anyone can run the program following the thread in **mdm_project.ipynb** file. There are 6 sections in code, Data load, Data preprocessing, Baselines, K-means with cosine similarity, Word2Vec, Word loud generation. **You can access the file in Google Colab here.**

Required Packages: Numpy, Pandas, nltk, re, itertools, sklearn, gensim, wordcloud.

References

[1] M. F. Porter, “An algorithm for suffix stripping,” *Program*, 1980.

[2] Q. Le and T. Mikolov, “Distributed representations of sentences and documents,” in *International conference on machine learning*, pp. 1188–1196, PMLR, 2014.

[3] Amueller, “Amueller/word_cloud: A little word cloud generator in python.”

Table 2: NMI Value Comparison of Different Clustering Methods with word2vec Vectors

	dim=64	dim=128	dim=256	dim=512
w2v-meanpooling-kmeans	0.3789	0.3550	0.3671	0.3892
w2v-tfidf-kmeans	0.6985	0.6989	0.6939	0.6891

A Further Attempt - Word2Vec

For a further exploration, we also test the performance of word2vec model[2]. We use the word2vec model integrated in *Gensim* Lib.

A.1 Data Preprocessing

We first split the titles and abstracts into sentences list, which sentence contains words. Next, we construct the similar pipeline used in Section 1. The word2vec model is trained on the processed corpus.

A.2 Methodology

After training, we get the word vectors in this corpus. However, since this is a document classification task, we are supposed to find a way to represent the document with the word vectors.

Here, we combined the TF-IDF scores with word vectors. Since the average pooling would cause the over-smoothness of critical information for a different word in a different document, we consider using the TF-IDF scores to do a weighted pooling of word vectors. The document vector of d_i would be calculated as follows:

$$repr_{d_i} = \frac{1}{k} \sum_{tf-idf(w_i)\geq 0} tf-idf(w_i) \cdot repr_{w_i} \tag{2}$$

, where k denotes the counts of words in document i whose $tf-idf$ score is non-zero.

A.3 Results

Table 2 shows the clustering results of word2vec+TF-IDF model. In the train of word2vec model, we use every word in the corpus for training and set the $window - size$ as 5. This combination of hyper-parameters outperforms others in our experiments.

We can see from Table. 2 that the weighted sum method is better than average pooling, however not outperform single TF-IDF compared with Table. 1. We conclude that the average of word vectors could more or less lose some key information for clustering. word2vec might not be a proper model here.

B Word Cloud of Clusters

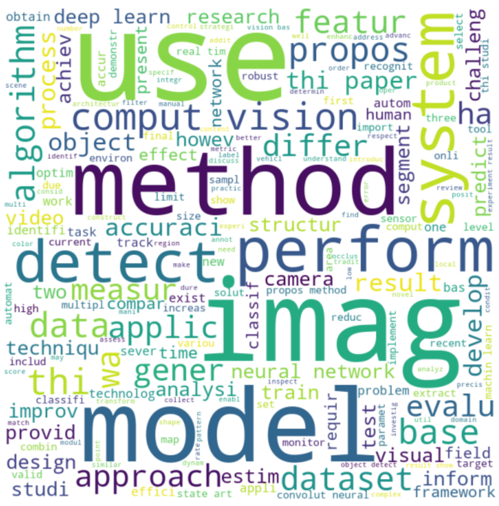

Figure B.1: Computer Vision

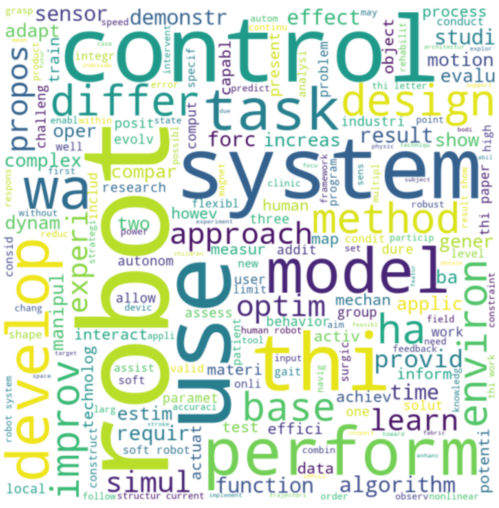

Figure B.2: Robot Design & Simulation

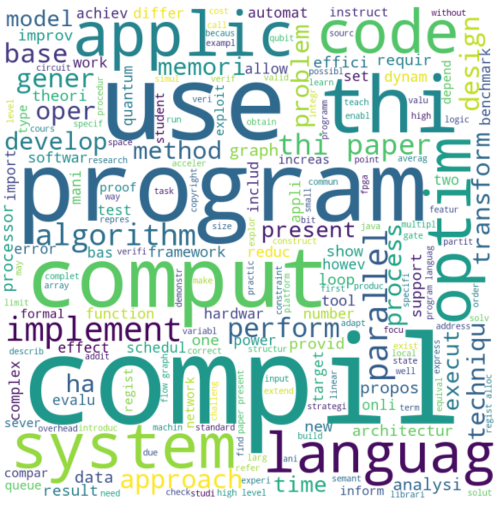

Figure B.3: Compiler Design

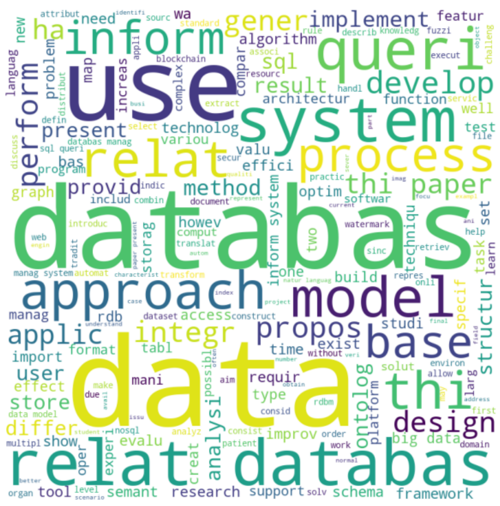

Figure B.4: Database

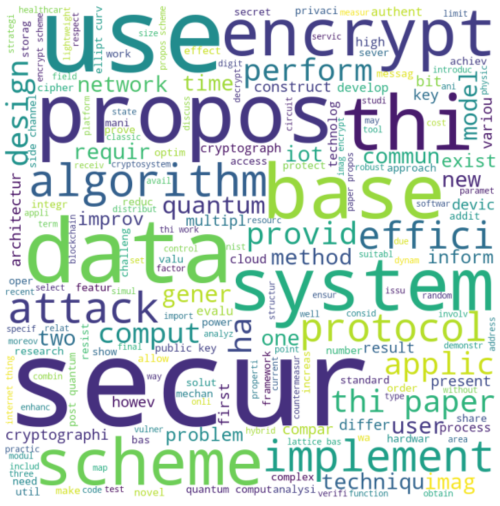

Figure B.5: Information Security & Cryptography