**Community Questions Answering - Question Comment Similarity with the use of Unsupervised ranking model**

(4 study points project work on Text-Based Information Retrieval)

by,

Marimuthu Ananthavelu (r0652832) Monika Filipčiková (r0683254)

KU Leuven.

***Summary***

This project work demonstrates the use of the Unsupervised model for ranking the comments with respect to a Question in the Community Questions Answering forums. The model which is used in this task is a Vector Space Model. The text pre-processing of the questions and comments were done with the help of available libraries for stop words, punctuations marks. The relevancy between each question and all the comments were referred as independent Vectors and their similarities were calculated with the help of Cosine scoring. The evaluation of model is done on the development set and compared with the baseline results and they are discussed. The output prediction file is generated for the ‘test\_input.xml’ is submitted for evaluation. The model is developed and evaluated using Python 3.5 version.

1. ***Introduction to the Project task:***

Community Questions Answering forums are helpful to know the information a user wants to know. The comments in this task are the different answers for a specific question which is presented the in the file with the .xml format. The objective of this task is to rank the very relevant answers to a question as higher and rank less to the least relevant answers.

1. ***Approach:***

The Intuition for choosing the Vector Space Model is because of the nature of Questions and Answers forum’s context in which the appearance of terms occurs. A sample Question may have an information need in the form of, *‘Where can I have the best coffee?’ a*nd It can have the relevant answer as: *‘The best coffee is in Block-200A’.*

The Vector Space model considers all the information needs and the relevant documents (comments in our case) in the Vector space. The chosen ranking model uses the same approach for finding the similarity between a question and a comment pair in each Thread in our task. An example of Vector space model representation shall be below:

*Whereas,*

Term-1

Question-2

Term-2

Comment-1

Terms are axes of the space,

Comments are vectors in this space.

Term-3

Question-1

**Fig-1: Vector space model representation**

In our task, we presented the model where the Question and all the comments were represented as Vectors. These vectors are weighted and normalised to each question and document pair and the dot product of those are used for calculating the Cosine score as follows:

1. For each question, the following are calculated:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Wordset | Question | | | | | |
| Term-Frequency | Term- weight frequency | Document- Frequency | Inverse document  frequency | Weight | Normalized  weights |
| Term-1 |  |  |  |  |  |  |
| Term-2 |  |  |  |  |  |  |
| … |  |  |  |  |  |  |

*Whereas,*

***Question =*** Mix of Question Category, Subject, and Body

***Wordset =*** Words which all are part of a single Thread (Question + all the comments in a Thread)

***term Frequency (tf-raw) =*** Number of times a term appears in a Question which produces the count Vector for each query or comment (document in the Information retrieval context) of the vocabulary size in a Thread.

This implies a comment with 5 occurrences of the term is more relevant than a comment with 1 occurrence of the term.

***Term weight frequency (tf-weight) =*** Converting the raw number of terms into weights in logarithmic scale as below:



Calculating the weights for a raw number of term frequencies reduces the linearity to sub-linear in. For an example,

0(tf)→ 0(tf-weight), 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.

***document-Frequency (df)=*** Number of comments a term appears (regardless of the number of times it occurs). The notion behind calculating this is, rare terms are more informative than frequent terms in a comment. Thus, giving high weights to the terms which are rare and the fewer weights for frequent terms (an inverse measure of informativeness).

***Inverse document frequency (idf) =*** Converted values of comments frequency using the total number of comments on a logarithmic scale.

idf = log (N / log (df))

*Whereas,*

N – Total number of comment

This can be seen as, if a word appears in every comment, does have a weight of ‘0’ due to the reason of non-discriminatory value between the comments.

**tf-idf weight term for query** = They are the product of tf-weight and idf-weight terms.

Normalisation = The product of tf-idf are normalised at the end.



1. For Comments, in a similar way, they are calculated for each comment (answer) as follows;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wordset | Comment\* | | | |
| Term-Frequency | Term- weight frequency | Weight | Normalized  weights |
| Term-1 |  |  |  |  |
| Term-2 |  |  |  |  |
| … |  |  |  |  |

\***There is no idf as it does not impact the scoring for a comment.**

The end dot product of the normalised weights between each question and comment pair can be calculated before they are summed to find the Cosine Score.



which provides the similarity of the two vectors i.e. between question and comment pair.

1. ***Model results discussion:***

The scores generated from the model on development set has been evaluated with the given relevancy file for the development set. These results are used for guiding towards generating the results for the given ‘test\_input.xml’ data with given question and comments. The predicted output file has been presented for the same as an enclosure.

While building the model, the following were given considerations for efficient implementation of functions and completeness

1. The Question subject, Question body and the question Category all merged into a text as single question representing the Query vector. This is for considering the possibility of having least information with either one of two of them. This also avoids having an empty query (which will make Vector Space model make scores as Zero).
2. The model ranks the comments as per the Cosine score for each question and comment pair.
3. Based upon each Question and comment pair score, the ranks are given from increasing order.

On the development set, the results look at the following way:

* 1. ***Functionality:***

The model is built in computing all the required information for finding the similarity using Cosine score. These functions are built into the model, which does not make use of any inbuilt library available in Python for doing the similar tasks. The functionalities are tested using a several custom possible test cases for ensuring the correct implementation.

* 1. ***Algorithmic Efficiency:***

***Time Efficiency:***

The implemented model comes with less or nil redundant code without making use of any inbuilt library in Python. This is also more scalable for a huge number of questions and comments with no additional efforts in implementation. This is clearly being seen and evident by having run the model overall ‘test\_input.xml’ which took less than 25 seconds computing time(with 2.2 Ghz processor).

The time efficiency of the implemented model is as O(N) which makes it more scalable for large scale for fair ranking implementations.

**Space Efficiency:**

Though the terms index per thread may create the sparse matrices in places because of the nature of some completely irrelevant comments, it is likely that majority of the comments bring the similar words with respect to the question which makes all the weights of comments more competitive in overall scoring.

* 1. ***Performance:***

The performance of the Vector Space model has been compared with other relevant ranking models for a similar application as below.

Implementation

Advantages of Vector Space Model:

* Vector space model uses simple computational framework for ranking
* It uses the concept of terms similarity between a Question and Comment for ranking the relevancy between the question and comment pair.

Improvements possibilities with the model:

* The semantics of natural language texts.

1. ***Enclosures:***

* Output file named ‘unsupervised\_rank\_marimuthu\_ananthavelu.txt’ for the given ‘test\_input.xml’.
* Model

- setup.py

- vsm.py

- processing.py

- processing\_word.py