Indian Institute of Information Technology Vadodara

CS 331 Information Retrieval

Mid Sem Lab Report

Group Member:

Yash Bhimani - 201951042

Devang Delvadiya - 201951055

Manthan Ghasadiya - 201951065

Rushi Javiya - 201951074

Course Co-ordinator:

Dr. Pratik Shah

CONTENTS

I. Experiment 4 : Vector Space Retrieval

Objective : Install and Quick hand on Elasticsearch and Kibana.

A. Introduction

The Vector-Space Model (VSM) for Information Retrieval represents documents and queries as vectors of weights. The vector space model is an algebraic representation of text documents (and other things) as vectors (such as index terms). It's utilised in information filtering, retrieval, indexing, and ranking relevancy. The SMART Information Retrieval System was the first to employ it.

In the Vector Space Model, Each document or query is a N-dimensional vector where N is the number of distinct terms over all the documents and queries.The i^{th} index of a vector contains the score of the i^{th} term for that vector. At retrieval time, the documents are ranked by the cosine of the angle between the document vectors and the query vector.

B. Cosine Similarity

To compute the similarity(closeness) between two vector,We calculate cosine angle between each document vector and the query vector. In order to compute the similarity between two vectors : d1(document 1), d2(document 2) the cosine similarity is used. Here also, To avoid the effect of document length as possible, the standard way of quantifying the similarity between two documents d1 and d2 is to compute the cosine similarity of their vector representations $\vec{V}(d1)$ and $\vec{V}(d2)$.[2]

$$sim(d1, d2) = \frac{\vec{V}(d1) \cdot \vec{V}(d2)}{\left|\vec{V}(d1)\right| \left|\vec{V}(d2)\right|} \quad (1)$$

C. Approach

Each document is represented as vector in vector space. We denote by $\vec{V}(d)$ the vector derived from document d, with one component in the vector for each dictionary term. The set of documents in a corpus then may be viewed as a set of vectors in a vector space, in which there is one axis for each term. To consider similarity between two vector(document) in this vector space is defined by cosine similarity. There is a far more compelling reason to represent documents as vectors : we can also view a query as a vector.

D. Why Elasticsearch/Kibana

This vector space retrieval can be done easily using elasticsearch and kibana. we have taken few examples to see their practical representation. We have taken English Hindi and Gujarati languages.[4]

E. Conclusion

we can briefly summarize that Elasticsearch is at its core a search engine, whose underlying architecture and components makes it fast and scalable. This can be used for many uses cases including search, analytics, and data processing and storage.Elasticsearch creates a data structure known as the Inverted Index which is primarily responsible for the lightning-fast search result retrieval. We first make index text analyzer and then analyze text using index. An index is the highest level entity that you can query against in Elasticsearch. Using Kibana We can analyze this all in real time.[1]

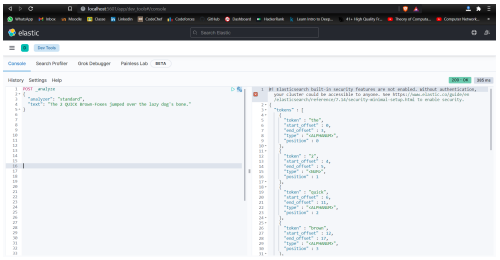


Fig. 1. Example of Defining the standard analyzers

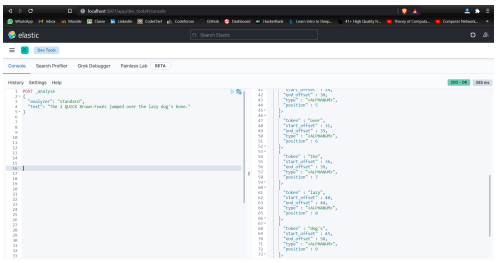


Fig. 2. Output of Defining the standard analyzers

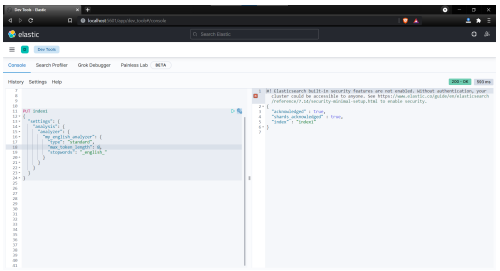


Fig. 3. Defining index1 in Elasticsearch and receiving acknowledgement

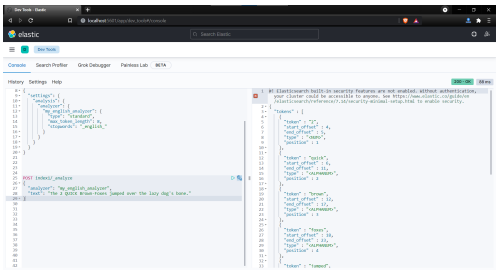


Fig. 4. Analyzing text using index and given text

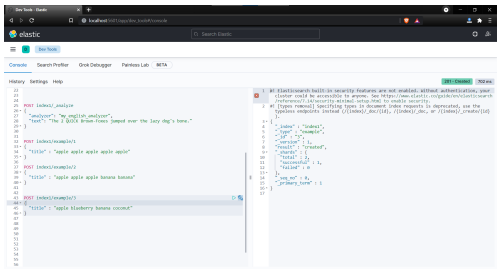


Fig. 5. Posting title of 3 queries into elasticsearch

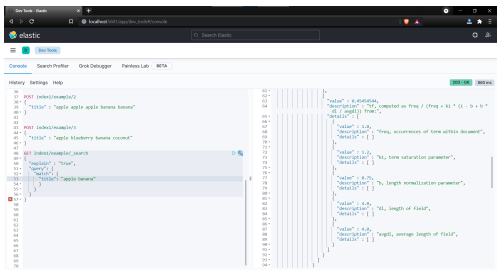


Fig. 6. Testing with different queries into elasticsearch

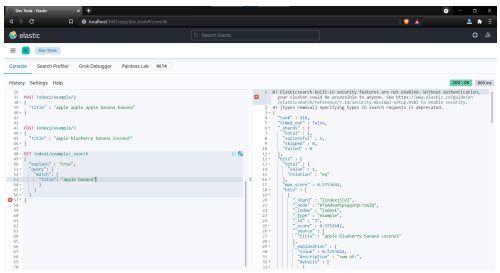


Fig. 7. Testing with different queries into elasticsearch

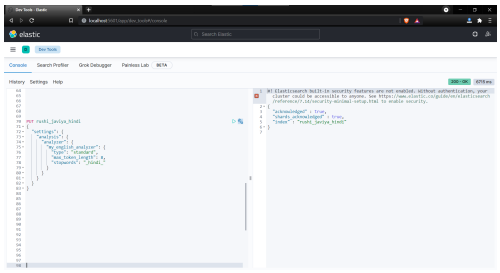


Fig. 8. Defining standard analyzer for Hindi language

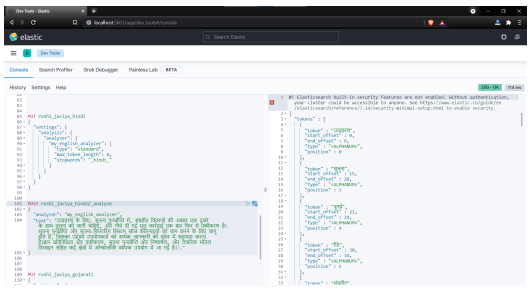


Fig. 9. Analyzing text using indexer and given text in Hindi

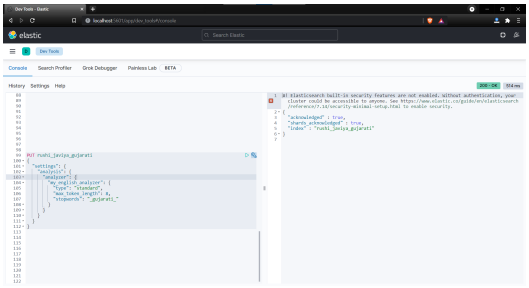


Fig. 10. Defining standard analyzer for Gujarati language

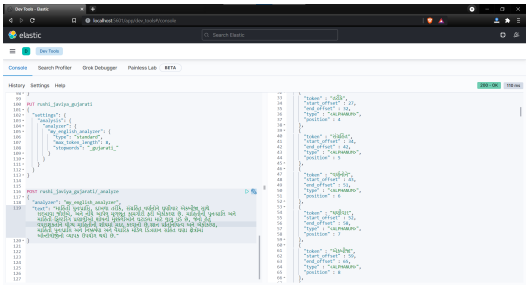


Fig. 11. Analyzing text using indexer and given text in Gujarati

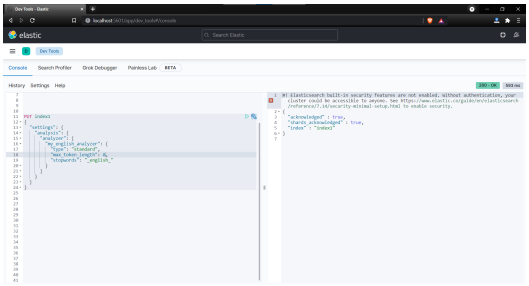


Fig. 12. Defining standard analyzer for English language

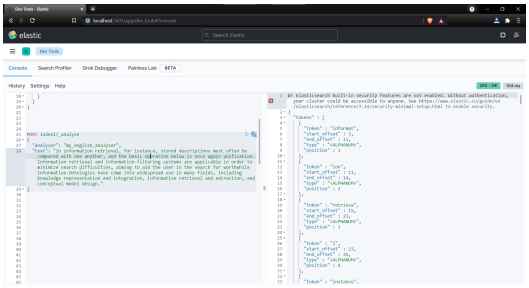


Fig. 13. Analyzing text using indexer and given text in English

II. LAB-5 | Text classification algorithms

A. Rocchio Algorithm

1) Introduction: For document vectors, there are three classifier models. The Rocchio model is the first of one in this. The Rocchio Algorithm is a well-known method of implementing relevance feedback. It simulates a method for combining relevance feedback data into a vector space model. The Rocchio feedback technique, was created with help of Vector Space Model. The algorithm assumes that most users have a broad idea of whether document should

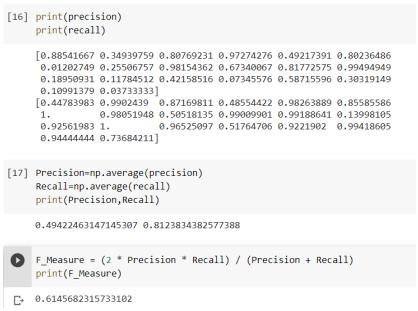


Fig. 14. Rocchio Classifier

be classified as relevant or non-related. As a result, the user's search query is modified to include an arbitrary percentage of relevant and non-related classes in order to improve the search engine's recall and, potentially, precision.

2) Code Explanation: We're trying to discover the vector space of label in terms of word id in this code. Pre-processing the data and retrieving information from the data is done first. After that, we create a numpy array with the number of labels divided by the number of different words. Now we'll fill in the word id versus label matrix that was produced earlier. Matrix created will be shown as below.

vector space	word$_i d1$	word$_i d2$	word$_i d3$
label$_1$	6	87838	877
label$_2$	7	78	5145
label$_3$	655	788	8507
label$_4$	545	18474	8560
label$_5$	88	878	6355

Now we normalize the the label vector.

Testing Now we have all the label vectors,. We will try to discover the absolute difference between all of them using the testing element. If the element is not present in the list then we have to find the label vector with the smallest difference. If an element does not appear in the list, it is ignored. The confusion matrix is now printed. Following that, we determine the document's precision and recall. After obtaining both precision and recall, we attempt to calculate the F score, which can be used to determine the Rocchio classifier model's accuracy.

3) Time Complexity of Rocchio Algorithm: The following table shows the time complexity for training and testing. Calculating the euclidean distance between a class centroid and the corresponding document can add complexity.

4) Limitation of Rocchio Algorithm: [9] When it comes to multi-modal classes and relationships, the Rocchio method frequently fails. It can miss-classify the similar objects. For example, despite having similar origins, the two inquiries "India" and "Pakistan" will seem significantly farther away in the vector space model.

B. KNN Algorithm

1) Introduction: The decision boundary is defined locally by the K nearest neighbour or KNN classification. K suggests the number of closest neighbours in this case. As a result, in order to work with KNN algorithm, we must calculate the distance between each new data point and training example. As a distance metric, we'll use the Euclidean distance algorithm. Our KNN Model selects K database items that are closest to the new data point. Now it chooses the high priority one, which implies that the most common label among those K entries will be the new data point's class. This process will repeat for all test datasets.

2) Code Explanation: In the first step, we have imported all of the necessary libraries and mounted the Google Drive with Colab. After that, we taken the news group dataset and combined all of the files into a single variable. We have taken the training and testing datasets and labels in the some variable as we move along. The data was then fitted into the vector variable after we initialised the tfidfvector in a single variable.

As we progress, we'll use the KNN classifier from the SkLearn library to initialise the variable. To check for any mistakes, we have used the cross validation method. Now we've arrived at the most important part of the code. We divided the data set into train and test parts in the test-classifier function. The train data set aids in the training of our model using the KNN classifier, whereas the test data set aids in the evolution of our model. We fit the training data set into the model and predicted the test data set in this final function.

As a result, now we are able to determine the model's accuracy, precision, recall, and f1 score for each document in the end. Also, we got the macro and weighted average of precision, recall and f1 score in the end.

3) Pros of KNN Algorithm: [7]

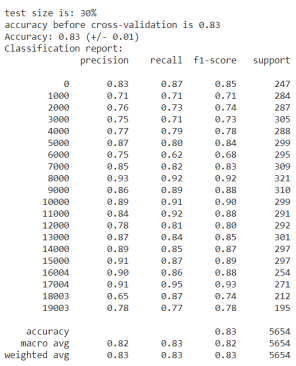
- We can train the model significantly faster with the K-nearest neighbour classifier than with conventional classification algorithms.
- The KNN approach can also be used to train a model with nonlinear data.
- The KNN algorithm is a simple, instance-based learning method.
- It can be applied to the regression issue.
- The average of the values of the k nearest neighbours is used to compute the object's output value.

4) Cons of KNN Algorithm:

- Because Euclidean distance is sensitive to magnitudes, the model's accuracy may be low
- When it comes to data storage, it demands a lot of memory compared to other classifier models.
- The KNN approach allows us to train the model more quickly, but the testing step is highly slow and requires more memory than other classifiers.
- The KNN approach cannot be used to train models with big data sets.

C. Naive Algorithm

Text classification is the process of classifying the texts. We usually classify them to help us grasp the various texts and their relevant classes. A Naive Bayes classifier is a

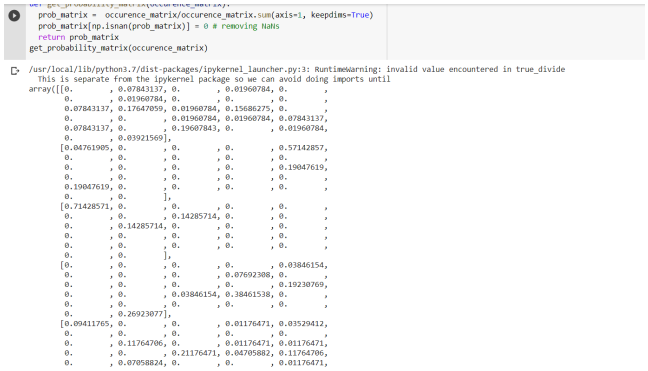


```
test size is: 30%
accuracy before cross-validation is 0.83
Accuracy: 0.83 (+/- 0.01)
Classification report:
              precision    recall  f1-score   support

           0       0.83      0.87      0.85       247
        1000       0.71      0.71      0.71       284
        2000       0.76      0.73      0.74       287
        3000       0.75      0.71      0.73       305
        4000       0.77      0.79      0.78       288
        5000       0.87      0.80      0.84       299
        6000       0.75      0.62      0.68       295
        7000       0.85      0.82      0.83       309
        8000       0.93      0.92      0.92       321
        9000       0.86      0.89      0.88       310
       10000       0.89      0.91      0.90       299
       11000       0.84      0.92      0.88       291
       12000       0.78      0.81      0.80       292
       13000       0.87      0.84      0.85       301
       14000       0.89      0.85      0.87       297
       15000       0.91      0.87      0.89       297
       16004       0.90      0.86      0.88       254
       17004       0.91      0.95      0.93       271
       18003       0.65      0.87      0.74       212
       19003       0.78      0.77      0.78       195

    accuracy                           0.83      5654
   macro avg       0.82      0.83      0.82      5654
weighted avg       0.83      0.83      0.83      5654
```

Fig. 15. KNN Classifier



```
  prob_matrix = occurence_matrix/occurence_matrix.sum(axis=1, keepdims=True)
  prob_matrix[np.isnan(prob_matrix)] = 0 # removing NaNs
  return prob_matrix
get_probability_matrix(occurence_matrix)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: invalid value encountered in true_divide
  This is separate from the ipykernel package so we can avoid doing imports until
array([[0.        , 0.07843137, 0.        , 0.01960784, 0.        ,
        0.        , 0.01960784, 0.        , 0.        , 0.        ,
        0.07843137, 0.17647059, 0.01960784, 0.15686275, 0.        ,
        0.        , 0.        , 0.01960784, 0.01960784, 0.07843137,
        0.07843137, 0.        , 0.19607843, 0.        , 0.01960784,
        0.        , 0.03921569],
       [0.04761905, 0.        , 0.        , 0.        , 0.57142857,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.19047619,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.19047619, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ],
       [0.71428571, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.14285714, 0.        , 0.        ,
        0.        , 0.14285714, 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.03846154,
        0.        , 0.        , 0.        , 0.07692308, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.19230769,
        0.        , 0.        , 0.03846154, 0.38461538, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.26923077],
       [0.09411765, 0.        , 0.        , 0.01176471, 0.03529412,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.11764706, 0.        , 0.01176471, 0.01176471,
        0.        , 0.        , 0.21176471, 0.04705882, 0.11764706,
        0.        , 0.07058824, 0.        , 0.        , 0.01176471,
```

Fig. 16. Naive Bayes Classifier

data classification technique based on Bayes' theorem. The Naive Bayes model is based on conditional probability of two occurrences occurring based on the odds of each individual event occurring.

1) How It Works: Texts are lengthy and contain a big amount of words. The number of times a word appears in a single text data set (i.e. term frequency-tf) and the number of times that word appears across all text data sets (i.e. inverse document frequency-idf) are used to construct feature vectors. The probabilities of appearing of the words in the text among the texts of a specific category are stored in feature vectors representing text, allowing the algorithm to calculate the likelihood of that text belonging to that category. Once we've determined the likelihoods of several categories, we'll look for the category with the highest probability, and we'll have done our text classification.

2) Code Explanation: To begin, import all the necessary libraries into the code and attach Google Drive to it. Following that, separate lists for document class, confidence score, and training labels are created. The relevant data is put in the label variable after importing the label file from the drive. Now data is imported, which consists primarily of words and their counts. We have completed the mapping procedure, and now we can focus on frequencies. As per the document, doc N will have a label at (N-1) location during the labelling procedure. Then, using the Theorem, we are able to determine the probability of each class.

3) Pros of Naive Algorithm: [8]

- Calculation time is small in comparision to other models.
- Capable to handle the problem of multiple-class prediction.
- Works very well with numerical input variables than category inputs.

4) Cons of Naive Bayes Algorithm:

- The naive Bayes assume that all features are independent which is not possible in the real life scenario.
- In some circumstances, its estimation may be incorrect.
- Zero frequency can be a problem for this model. Due to which smoothing technique is used for correctness.

D. Conclusion

- f1 score of KNN classifier : 0.83
- f1 score of Rocchio classifier : 0.61
- f1 score of Naive Bayes classifier : 0.78

Knn is the strongest text classifier among these three models, according to the f1 score.

link to code folder

III. LAB-2 | Block Sort Based Indexer and Building Term-Document Matrix (TF-IDF)

A. Block Sort Based Indexer

1) Introduction: We begin by traversing the collection and assembling all term–docID pairs. The pairs are then sorted with the term as the primary key and the docID as the secondary key. Finally, we organise each term's docIDs into a postings list and compute statistics such as term and document frequency. All of this may be done in memory for small collections.[6]

2) Process: We represent words as termIDs, which are unique serial numbers, to make index construction more efficient. We can either create the inverted index in the first run and then build the mapping from terms to termIDs on the fly while processing the collection, or we can do both in a two-pass method.

We must will use an external sorting method, i.e. one that requires disc, because main RAM is inadequate. The essential condition of such an algorithm for reasonable performance is that it reduce the amount of random disc seeks during sorting — sequential disc reads are far quicker than seeks.The blocked sort-based indexing algorithm, or BSBI in Figure 4.2 of code, is one solution. BSBI (i)divides the collection into equal-sized parts, (ii) sorts the termID–docID pairings in memory, (iii) saves intermediate sorted results to disc, and (iv) combines all intermediate results into the final index.

BSBINDEXCONSTRUCTION()
1 n ← 0
2 while (all documents have not been processed)
3 do n ← n +1
4 block ← PARSENEXTBLOCK()
5 BSBI-INVERT(block)
6 WRITEBLOCKTODISK(block, fn)
7 MERGEBLOCKS(f1, . . . , fn; fmerged)
Figure 4.2 of code Blocked sort-based indexing. The algorithm stores inverted blocks in files f1, . . . , fn and the merged index in fmerged.

3) How does it work: Documents are divided into termID–docID pairs and stored in memory until a fixed-size block (PARSENEXTBLOCK in Figure 4.2 of code) is full. We set a block size that fits perfectly within memory to enable a rapid in-memory sort. The block is then written on disc in reverse order. Inversion consists of two phases. To begin, we sort the termID–docID pairs. Then, in a posts list, POSTING, we gather all termID–docID pairings that have the same termID. A posting is nothing more than a docID. The outcome is saved to disc as an inverted index for the block we just read. In the last phase, the approach merges the ten blocks into a single massive combined index.

B. Term-Document Matrix(TF-IDF)

1) Introduction: Tf-idf is available in a variety of flavours, some more complicated than others. The basic mathematical operations to learn are addition, multiplication, and division. At some point, we'll need to calculate the natural logarithm of a variable, but most online calculators and calculator mobile apps can do it for us. The raw term counts for the first thirty words of Bly's obituary are listed below in alphabetical order, but this version contains a second column indicating the number of periodicals in which each phrase may be found.

2) Process: The document frequency (df) is a tally of how many times each word appears in the corpus. (Document frequency for a specific word is denoted by dfi.)

N/dfi is the simplest formula for determining inverse document frequency for each term, where N is the total number of documents in the corpus. Many implementations, however, require extra processes to normalise the results. In TF-IDF, normalisation is used in two ways: first, to prevent term frequency bias from words in shorter or longer documents, and second, to calculate the idf value for each word (inverse document frequency). The method in Scikit-Learn, for example, represents N as N+1, calculates the natural logarithm of (N+1)/dfi, and then adds 1 to the final result. In the section below labelled "Scikit-Learn Settings," we'll return to the concept of normalisation.

The following equation can be used to express Scikit-idf Learn's transformation:

idfi = In[(N+1)/dfi] +1

After calculating idf;, tf-idf; is tf multiplied by idf.

tf-idfi = tfi x idfi

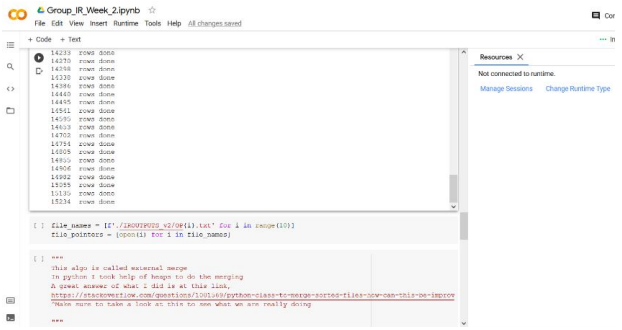


Fig. 17. Processing database and creating block sorted based index files

If you've worked with them before, they can provide a more lucid overview of an algorithm's processes than any well-written prose. I've added two new columns to the previous terms frequency table to make the idf and tf-idf equations more obvious. To calculate the final tf-idf score, the first new column represents the derived idf score, while the second new column multiplies the Count and Idf columns.

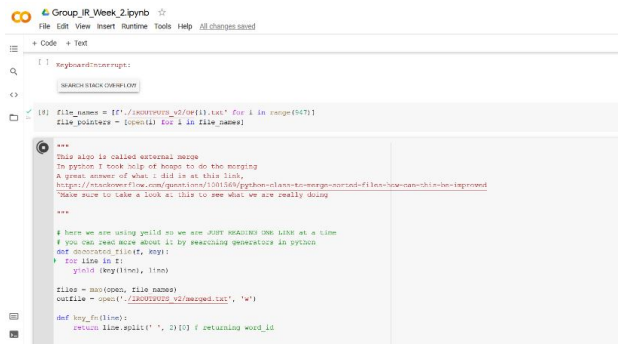


Fig. 18. Merging the sorted files in one

C. Conclusion

It's worth noting that the idf score is higher if the phrase appears in fewer papers, although the visible idf score ranges from 1 to 6. Different scales would result from different normalisation approaches.
Link to code folder

IV. LAB-6 | Latent Semantic Indexing

A. Introduction

Latent semantic indexing (also known as Latent Semantic Analysis) is a way of evaluating a series of documents to find statistical co-occurrences of words that appear together, which subsequently provides information about the themes of those words and documents.[5]

Two well-studied machine learning algorithms to text classification are Support Vector Machines (SVM) and k-nearest neighbours (kNN). They're both used on a vector space model that's defined over a collection of words (BOW). Documents are represented as vectors across a set of dimensions in this approach, with each dimension representing to a word in the BOW.

B. Processing

LSI's goal is to extract a smaller number of dimensions that are more reliable markers of meaning than single phrases. To get at these latent dimensions, LSI employs the Singular Value Decomposition (SVD). In theory, SVD does a two-mode factor analysis, placing both terms and documents in a single space specified by the extracted dimensions. The original term document matrix is broken down into three matrices by SVD, two of which indicate the altered representations of words and documents in terms of the additional dimensions, while the third assigns "weights" to these dimensions. The theory underpinning LSI is that due to word-choice unpredictability, less important dimensions correspond to "noise." By removing these noisy dimensions, a lower rank approximation to the original matrix is created. This approximation is a smoothed (blurred) version of the original, and it is supposed to reflect relationships between terms and documents more precisely in terms of the underlying concepts. The generated approximation is the closest matrix of its rank to the original in the least-squares sense, which is an intriguing property of SVD.[10] While LSI has proven to be effective in retrieval tasks, it has several drawbacks when it comes to categorization. As previously stated, because LSI is blind to training document class labels, the extracted dimensions are not always the best in terms of class discrimination. In addition, LSI may filter out rare terms with strong discriminatory power.

- SVD minimises the "distance" between the two matrices as measured by the 2-norm by representing A in a reduced dimensional space:
$$\triangle = \|A - A1\|_2$$
- The SVD projection is computed by decomposing the document-by-term matrix At×d into the product of three matrices, Tt×n , Sn×n , Dd×n
$$A_{t*d} = T_{t*n} S_{n*n} (D_{d*n})^T$$

- where t is the number of terms, d is the number of documents, n = min(t,d), T and D have orthonormal columns, i.e. $TT^T = D^T D = I, rank(A) = r, S = diag(\sigma_1, \sigma_2, ..., \sigma_n,), \sigma_i > 0 for 1 \leq i \leq r, \sigma_j = 0 for j \geq r+1$.

C. Queries

- For purposes of information retrieval, a user's query must be represented as a vector in k-dimensional space and compared to documents. A query (like a document) is a set of words. For example, the user query can be represented by

$$\hat{q} = q^T T_{t*k} S_{k*k}^{-1}$$

- where q is simply the vector of words in the users query, multiplied by the appropriate term weights. The sum of these k-dimensional terms vectors is reflected by the term $q^T T_{t*k}$ in the above equation, and the right multiplication by S_{k*k}^{-1} differentially weights the separate dimensions.

D. Updating

- Updating refers to the general process of adding new terms and/or documents to an existing LSI-generated database. Updating can mean either folding-in or SVD-updating.
- Folding-in terms or documents is a much simpler alternative that uses an existing SVD to represent new information.
- Recomputing the SVD is not an updating method, but a way of creating an LSI-generated database with new terms and/or documents from scratch which can be compared to either updating method.

The equation for folding documents into the space can again be derived from the basic SVD equation:

$$A = TSD^T$$
$$T^T A = T^T TSD^T$$
$$T^T A = SD^T$$

E. Lab Practical

link to code

F. Conclusion

From the above experiment we can conclude some advantages of this model.[3]

1. The Assumption in LSI is that the new dimensions are a better representation of documents and queries.

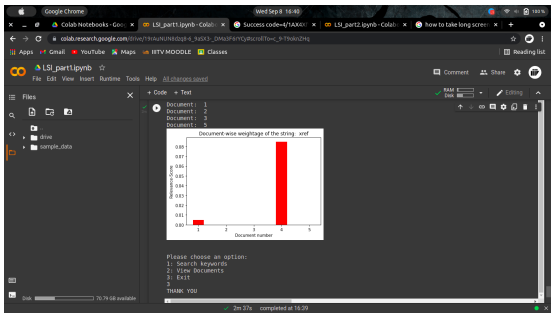


Fig. 19. 1

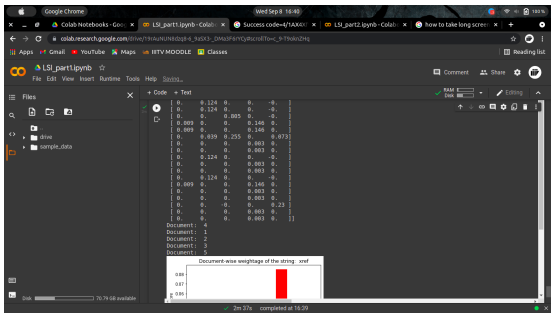


Fig. 20. 2

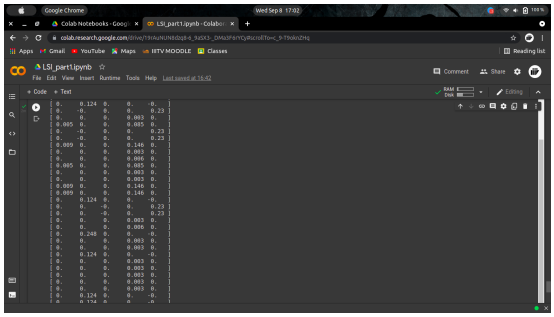


Fig. 21. 3

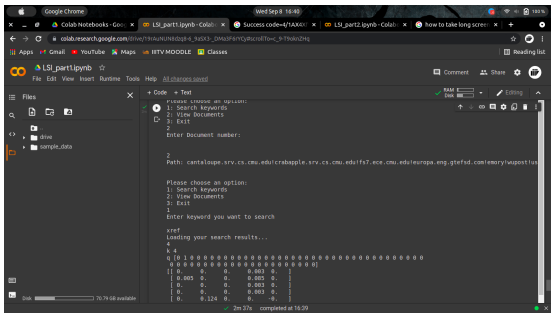


Fig. 22. 4

2. The same underlying concept can be described using different terms.

3. Polysemy describes the availability of the words that have more than one meaning, which is common property of language.

This model also have some disadvantages as following

1. It becomes very challenging and time-consuming to handle very much bigger matrix and it becomes exponentially more time consuming with the increment of unique words.

2. we can say that the SVD representation is more dense. Many documents have more than 200 unique terms. So the sparse vector representation will take up more storage space than the compact SVD representation if we reduce to 200 dimensions.

REFERENCES

[1] Ralf Abueg. *Elasticsearch: What It Is, How It Works, And What It's Used For*. URL: https://www.knowi.com/blog/what-is-elastic-search/.

[2] AngeloCatalani. *Web Information Retrieval | Vector Space Model*. URL: https://www.geeksforgeeks.org/web-information-retrieval-vector-space-model/.

[3] Raghavendran Balu. *ADV*. URL: https://www.quora.com/What-are-the-advantages-and-disadvantages-of-Latent-Semantic-Analysis.

[4] Giovanni Pagano Dritto. *An Overview on Elasticsearch and its usage*. URL: https://towardsdatascience.com/an-overview-on-elasticsearch-and-its-usage-e26df1d1d24a.

[5] Susan li. *LSI defination*. URL: https://towardsdatascience.com/latent-semantic-analysis-sentiment-classification-with-python-5f657346f6a3.

[6] Christopher D. Manning. *block sort based indexing*. URL: https://nlp.stanford.edu/IR-book/html/htmledition/blocked-sort-based-indexing-1.html.

[7] Avinash Navlani. *KNN*. URL: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn.

[8] Pavan Vadapalli. *Naive Bayes*. URL: https://www.upgrad.com/blog/naive-bayes-explained/.

[9] wikipedia. *Rocchio Algorithm*. URL: https://en.wikipedia.org/wiki/Rocchio_algorithm.

[10] wikipedia. *SVD*. URL: https://en.wikipedia.org/wiki/Singular_value_decomposition.