CS6370 Team 6 Project report*

Durga Sandeep [ME16B125] and Sureddy Abhishek [ME16B166]

IIT Madras
me16b166@smail.iitm.ac.in
me16b125@smail.iitm.ac.in

Abstract. In this report, we make hypotheses to address some of the limitations of vector space model and then discuss the observations made. We also discuss about some methods, that can significantly reduce search time like Clustering, LDA(Topic modeling). Methods which can improve user experience like : Query auto-completion (using text generation using LSTMs), It includes models like LSA (Latent Semantic Analysis), Query expansion also. we will also discuss some of methods we have tried to implement, but couldn't implement (because of technical problems.). Here, we use Vector space model as the baseline model against which all comparisons are made.

Keywords: LDA · Clustering · Query auto-completion · LSA · Query expansion · baseline model.

1 Introduction

To evaluate any model, we should first define a baseline model against which our model has to be compared in all the aspects, which the user is interested in. As a part of that, we have implemented two versions of baseline models and used the second one (baseline model 2) to compare various evaluation parameters against the other implemented models.
We use **cosine similarity** as a measure to rank the documents in all models. Evaluation parameters include **MAP (Mean Average Precision), Recall, f-score and n-DCG score.** In this section, we will discuss more about the details of Baseline models. Then we'll compare both of them.

1.1 Baseline model 1

A vector space model created using the tf-idf scores of words in the documents. The steps involved in the creation of this model are:

1. **Tokenization :** Making each document into a list of sentences. Making each sentence a list of words.
2. **Stopword removal :** Now stopwords (i.e. most common words that occur in almost all documents, like a,an,the,...) are removed from this sub sub list of words.
3. **Inflection reduction :** Now we use **lemmatization** to reduce the all words to their corresponding root words. we didn't use stemming because, it's a crude way of extracting root word. (benifits of lemmatization over stemming are mentioned here [1])
4. **TF-IDF matrix :** A tf-idf matrix is constructed and then each column is normalized to unit length for the documents. For queries, only a term matrix is constructed.
5. **Ranking :** We use cosine similarity as a measure of ranking the documents, for each query.

1.2 Baseline model 2

Here, we have done some basic text-preprocessing, which gave an increment in all evaluation measures in comparison with baseline model 1. The details of text preprocessing steps involved in the model are:

1. **Converting all letters to lowercase :** In the given corpus, there are no uppercase letters but given query can have uppercase letters. So in general, we are making all letters lowercase. This helps when the dataset has uppercase also. For example - Aeroplane, aeroplane both are same words but if you do not convert them to lowercase, then the bag of words model treats both words differently which in turn increase our dimensionality of vector("Curse of dimensionality")
2. **Removing numbers in the text :** Usually we might have words attached with numbers as well. So in this case we are removing all the numbers in the corpus
3. **Removing punctuations, accent marks and other diacritics.** This is very important. If you don't remove it in the corpus, then the bag of words model considers the punctuation as one more word and adds to the dimension. For example - if i have a sentence ["Hi, I am a Robot."] After tokenization it gives ['hi', ',', 'i','am','a','robot',"."]. This can lead to a very poor model as it increases the vocabulary size i.e dimension of each vector.

* Team 6.

4. **Removing extra white spaces :** self explanatory.
5. **Removing Stopwords :** Here we remove standard stop words provided by nltk library and also very less frequent words which are corresponding to the corpus.
6. **Stemming or Lemmatization :** Stemming is a process of reducing words to their word stem, base or root form. The aim of lemmatization, like stemming, is to reduce inflectional forms to a common base form. As opposed to stemming, lemmatization does not simply chop off inflections. Instead it uses lexical knowledge bases to get the correct base forms of words.

Now each document is represented as a vector constructed using tf-idf scores. The ranking is the same as in baseline model 1.

1.3 Baseline model 1 Vs Baseline model 2

The comparison table for both the models is shown in below table 1

	Base line model 1					Baseline model 2				
k	**MAP**	**Recall**	**f-score**	**n-DCG**	**Precision**	**MAP**	**Recall**	**f-score**	**n-DCG**	**Precision**
1	0.636	0.11	0.181	0.493	0.636	0.693	0.12	0.193	0.53	0.693
2	0.695	0.18	0.26	0.40	0.538	0.733	0.188	0.266	0.412	0.567
3	0.699	0.23	0.291	0.372	0.467	0.734	0.241	0.302	0.390	0.501
4	0.695	0.264	0.303	0.364	0.411	0.732	0.287	0.328	0.390	0.461
5	0.685	0.294	0.309	0.364	0.372	0.721	0.313	0.328	0.391	0.412
6	0.677	0..323	0.314	0.372	0.343	0.710	0.342	0.331	0.396	0.379
7	0.668	0.347	0.315	0.376	0.320	0.697	0.368	0.334	0.402	0.352
8	0.658	0.364	0.311	0.383	0.301	0.687	0.389	0.330	0.407	0.332
9	0.653	0.382	0.310	0.388	0.285	0.680	0.406	0.325	0.411	0.311
10	0.644	0.40	0.304	0.393	0.267	0.674	0.419	0.318	0.414	0.291

Table 1. Baseline model1 Vs Baseline model2

We can clearly see that model 2, outperforms model 1 in all aspects (by 3 to 5 %). The vocabulary in model 1 was around 8000 words, where as in model 2, it was just 5000 words. This is because of proper text preprocessing which was done in model 2. (We have noticed some punctuation left, without being removed in model 1, which results in poor similarity scores, thereby reducing the model performance).

The general trends of the evaluation measures for all models, were as shown in figure 1

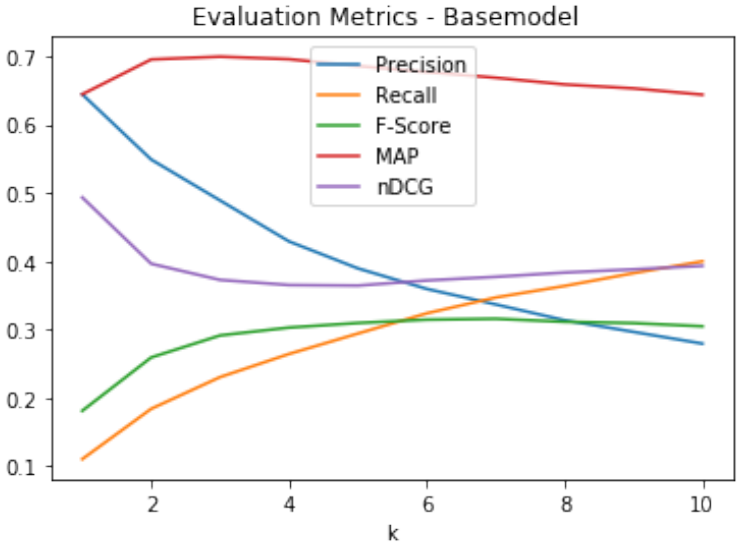

Fig. 1. General trend of all evaluation measures

More information about the trends is already discussed in assignment 2 report.
From now onwards, we use Baseline model 2 as our Base model.

2 Limitations of Vector Space Model :

The following are limitations of vector space model, which we are trying to address :

1. **(Curse of Dimensionality)** Long documents are poorly represented because they have poor similarity values (a small scalar product and a large dimensionality)
2. Vector space model assumes the orthogonality of dimensions (words), but it is not true in general.
3. Semantic sensitivity. Documents with similar context but different term vocabulary won't be associated.
4. The order in which the terms appear in the document is lost in the vector space representation.

From next section onwards we will see various hypotheses to solve these problems.

3 Document Representation & Orthogonality(limitation 1 & 2):

Hypothesis :
Reducing the dimension of vector representation for all documents by using LSA for better representation in terms of space(storage of huge vectors).LSA solves the problem of orthogonality of dimensions to an extent i.e mitigates the problem of identifying synonymy(merge the dimensions associated with terms that have similar meanings), by capturing the first order associations between the words.

3.1 LSA

Latent semantic analysis (LSA) is a technique in NLP, in particular distributional semantics, of analyzing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms.[2]
There could be various reasons for these approximations:

1. The original term-document matrix is presumed too large for the computing resources; in this case, the approximated low rank matrix is interpreted as an approximation.
2. The original term-document matrix is presumed noisy: for example, anecdotal instances of terms are to be eliminated. From this point of view, the approximated matrix is interpreted as a de-noisified matrix (a better matrix than the original)
3. The original term-document matrix is presumed overly sparse relative to the "true" term-document matrix. That is, the original matrix lists only the words actually in each document, whereas we might be interested in all words related to each document—generally a much larger set due to synonymy.

Approach:
We have implemented two versions of LSA :

1. LSA (corpus : cranfield docs) on TF-IDF matrix.
2. LSA (corpus : cranfield docs & brown corpus) on TF-IDF matrix.

For both versions of LSA, the best value of evaluation measures were found around $k = 500$ (latent dimensions.)

3.2 Comparison with Baseline model

The comparison of LSA model 1 with base model 2 is shown in the table 2
The comparison of LSA model 2 (LSA with large corpus) is shown in the table [3]

3.3 Evaluation of Hypothesis

3.4 Inference

Document Representation can be improved using Latent Semantic Analysis (The most important drawback of LSA is the extreme computational effort behind SVD. The computational complexity of this algorithm is $O(n^2 * k^3)$, where n is the number of documents plus the number of terms, and k is the number of embedded latent dimensions.)

4 Addressing Semantic Sensitivity(Limitation No. 3):

Hypothesis :
Different words can have the same meaning called synonyms. So using Query Expansion we would like to get the synonyms using knowledge structure.

	LSA with 500 latent dimensions					Base model 2				
k	**Precision**	**Recall**	**f-score**	**MAP**	**n-DCG**	**Precision**	**Recall**	**f-score**	**MAP**	**n-DCG**
1	0.707	0.119	0.195	0.707	0.532	0.693	0.12	0.193	0.693	0.53
2	0.578	0.19	0.269	0.738	0.414	0.567	0.188	0.266	0.733	0.412
3	0.513	0.244	0.307	0.741	0.394	0.501	0.241	0.302	0.734	0.39
4	0.471	0.294	0.335	0.733	0.397	0.461	0.287	0.328	0.732	0.39
5	0.418	0.317	0.333	0.728	0.395	0.412	0.313	0.328	0.721	0.391
6	0.383	0.346	0.335	0.718	0.4	0.379	0.342	0.331	0.71	0.396
7	0.363	0.378	0.342	0.701	0.409	0.352	0.368	0.334	0.697	0.402
8	0.34	0.399	0.339	0.691	0.413	0.332	0.389	0.33	0.687	0.407
9	0.32	0.418	0.336	0.679	0.417	0.311	0.406	0.325	0.68	0.411
10	0.302	0.433	0.33	0.67	0.422	0.291	0.419	0.318	0.674	0.414

Table 2. LSA 1 Vs Baseline model2

	LSA with brown corpus (500 latent dim)					Base model 2				
k	**Precision**	**Recall**	**f-score**	**MAP**	**n-DCG**	**Precision**	**Recall**	**f-score**	**MAP**	**n-DCG**
1	0.702	0.117	0.192	0.702	0.532	0.693	0.12	0.193	0.693	0.53
2	0.573	0.186	0.264	0.733	0.414	0.567	0.188	0.266	0.733	0.412
3	0.511	0.244	0.307	0.738	0.393	0.501	0.241	0.302	0.734	0.39
4	0.458	0.285	0.325	0.733	0.392	0.461	0.287	0.328	0.732	0.39
5	0.42	0.319	0.334	0.716	0.392	0.412	0.313	0.328	0.721	0.391
6	0.379	0.341	0.331	0.71	0.393	0.379	0.342	0.331	0.71	0.396
7	0.357	0.372	0.336	0.696	0.403	0.352	0.368	0.334	0.697	0.402
8	0.336	0.393	0.334	0.684	0.409	0.332	0.389	0.33	0.687	0.407
9	0.315	0.409	0.329	0.673	0.414	0.311	0.406	0.325	0.68	0.411
10	0.298	0.427	0.325	0.662	0.418	0.291	0.419	0.318	0.674	0.414

Table 3. LSA 2 Vs Baseline model2

4.1 Query Expansion model

Query Expansion broadens the query by introducing additional tokens or phrases. The search engine automatically rewrites the query to include them. For example, the query vp marketing becomes **(vp OR "vice president")** marketing.The implementation details are :

1. We train a **continuous bag of words(CBOW)**[5] model on the corpus and produce vector representation of all words in the corpus. These word vectors are also known as word embeddings.
2. For every word in the query, we'll take the top most similar word (cosine similarity), matching with the query word and append it to the query.
   This can be illustrated for a word as follows in figure 2

```
[282]:  1 res = np.array(model.wv.most_similar(positive=["good"]))[:,0].tolist()
        2 res

[282]: ['agreement',
        'result',
        'experimental',
        'comparison',
        'theoretical',
        'data',
        'compare',
        'satisfactory',
        'prediction',
        'reasonably']
```

Fig. 2. query expansion model predicting similar words to "good".

3. Now we take the query's vector representation and rank the documents as per cosine similarity values.

4.2 Comparing with baseline model

The comparison of Query Expansion model is shown in the table [4]

	Query expansion model					Base model 2				
k	**Precision**	**Recall**	**f-score**	**MAP**	**n-DCG**	**Precision**	**Recall**	**f-score**	**MAP**	**n-DCG**
1	0.568	0.0968	0.158	0.568	0.432	0.693	0.12	0.193	0.693	0.53
2	0.449	0.145	0.207	0.609	0.333	0.567	0.188	0.266	0.733	0.412
3	0.394	0.188	0.24	0.622	0.319	0.501	0.241	0.302	0.734	0.39
4	0.358	0.220	0.258	0.621	0.317	0.461	0.287	0.328	0.732	0.39
5	0.334	0.253	0.266	0.616	0.321	0.412	0.313	0.328	0.721	0.391
6	0.312	0.281	0.272	0.605	0.328	0.379	0.342	0.331	0.71	0.396
7	0.29	0.303	0.273	0.598	0.33	0.352	0.368	0.334	0.697	0.402
8	0.275	0.322	0.274	0.585	0.333	0.332	0.389	0.33	0.687	0.407
9	0.262	0.340	0.275	0.578	0.340	0.311	0.406	0.325	0.68	0.411
10	0.247	0.345	0.270	0.570	0.345	0.291	0.419	0.318	0.674	0.414

Table 4. Query expansion model Vs Basemodel 2

4.3 Evaluation of Hypothesis

5 Methods implemented to reduce search time

As the number of documents to search increase, the time taken to retrieve relevant documents to a particular query also increases linearly. So we came up with two methods to reduce the search time required to retrieve relevant documents :

1. Clustering of documents.
2. Topic modeling using LDA.

5.1 Clustering of documents

Clustering is an un-supervised method of grouping data. Here, we use K-means clustering[3]. (an application of EM algorithm). The detailed steps of implementation are :

1. Apply KMeans clustering on the TF-IDF matrix of documents, and cluster them into 'k' clusters. The value of k can be found using elbow plot. The best value was found to be around 6. See figure [3] and [4] :

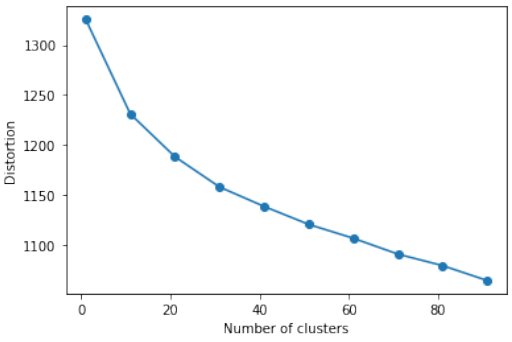

Fig. 3. Elbow plot to find optimal clusters

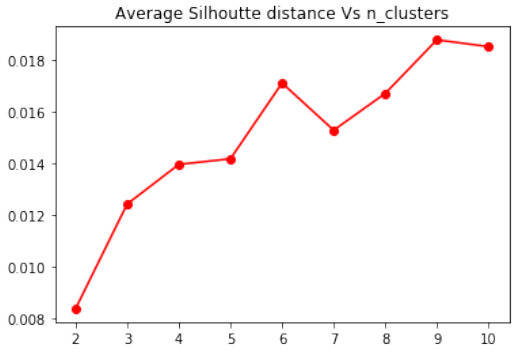

Fig. 4. Avg Silhoutte distance

2. Now for a given query, we find the cosine similarity between the query and k cluster centres. Based on the cosine similarity scores, we decide on which cluster the query belongs to.
3. Now, we take cosine similarity of a query with all documents in that cluster and rank those documents.

	Clustering, k = 6	**Base model**
Avg retrieval time	3.7 ms	11 ms

Table 5. Retrieval time comparison

5.2 Comparison of Clustering model with Base model

Comparison interms of Time of retrieval :
Comparing Clustering model with Base model, in terms of average time of retrieval is shown in table [5]
We see that, the **retrieval time is reduced by a factor of 3**. This seems to be a good improvement.
Comparison interms of evaluation metrics
The comparison of clustering model with base line is shown in table [6] :

	Clustering, k = 6					**Base model 2**				
k	**Precision**	**Recall**	**f-score**	**MAP**	**n-DCG**	**Precision**	**Recall**	**f-score**	**MAP**	**n-DCG**
1	0.573	0.093	0.154	0.573	0.412	0.693	0.12	0.193	0.693	0.53
2	0.458	0.145	0.207	0.609	0.318	0.567	0.188	0.266	0.733	0.412
3	0.407	0.188	0.24	0.611	0.302	0.501	0.241	0.302	0.734	0.39
4	0.369	0.222	0.258	0.602	0.299	0.461	0.287	0.328	0.732	0.39
5	0.34	0.249	0.266	0.597	0.303	0.412	0.313	0.328	0.721	0.391
6	0.307	0.267	0.264	0.594	0.304	0.379	0.342	0.331	0.71	0.396
7	0.29	0.291	0.268	0.583	0.31	0.352	0.368	0.334	0.697	0.402
8	0.27	0.305	0.265	0.576	0.313	0.332	0.389	0.33	0.687	0.407
9	0.253	0.319	0.261	0.57	0.317	0.311	0.406	0.325	0.68	0.411
10	0.237	0.328	0.255	0.562	0.319	0.291	0.419	0.318	0.674	0.414

Table 6. Clustering model Vs Baseline model2

There is a decrement in all evaluation measures by approximately 0.1 . The decrement is expected, because, we have reduced our search space. However, if speed of retrieval is more important, then one can use this method.

5.3 Topic modeling using LDA

Topic modeling is an unsupervised machine learning technique that automatically analyzes text data to determine cluster words for a set of documents. More about this can be found here[6].
under lying assumptions of LDA (Latent Dirchlet Allocation) is same as that of LSA.(i.e. similar topics make use of similar words). The main difference between LSA and LDA is that LDA assumes that the distribution of topics in a document and the distribution of words in topics are Dirichlet distributions.
There are two hyperparameters that control document and topic similarity, known as alpha and beta, respectively. A low value of alpha will assign fewer topics to each document whereas a high value of alpha will have the opposite effect. A low value of beta will use fewer words to model a topic whereas a high value will use more words, thus making topics more similar between them. The output of algorithm is a vector that contains the coverage of every topic for the document being modeled. For more information on computing these probabilities, we can refer to the original LDA paper[7]. Model implementations steps are:

1. Documents are grouped, based on similar topics.
2. Now, we will compute the centroids of all topics. (centroid based classifier[8]). Then we compute cosine similarity of these centroids with given query vector.
3. Only the documents belonging to that topic would be considered for ranking.

comparison with base model Comparing the LDA model, with baseline model in table [7] :

6 Query auto-completion

To improve user interaction with search engine. we can model incomplete queries, or this can be further incorporated with browsers, so that they can provide auto-completion of a query that we type, as shown in figure5

	LDA model					Base model 2				
k	**Precision**	**Recall**	**f-score**	**MAP**	**n-DCG**	**Precision**	**Recall**	**f-score**	**MAP**	**n-DCG**
1	0.484	0.085	0.139	0.484	0.338	0.693	0.12	0.193	0.693	0.53
2	0.407	0.133	0.188	0.547	0.277	0.567	0.188	0.266	0.733	0.412
3	0.367	0.178	0.223	0.564	0.271	0.501	0.241	0.302	0.734	0.39
4	0.337	0.209	0.239	0.567	0.276	0.461	0.287	0.328	0.732	0.39
5	0.307	0.237	0.247	0.562	0.275	0.412	0.313	0.328	0.721	0.391
6	0.279	0.257	0.247	0.551	0.278	0.379	0.342	0.331	0.71	0.396
7	0.255	0.27	0.241	0.546	0.282	0.352	0.368	0.334	0.697	0.402
8	0.24	0.286	0.241	0.542	0.285	0.332	0.389	0.33	0.687	0.407
9	0.229	0.302	0.24	0.536	0.29	0.311	0.406	0.325	0.68	0.411
10	0.215	0.312	0.235	0.533	0.295	0.291	0.419	0.318	0.674	0.414

Table 7. LDA model Vs Baseline

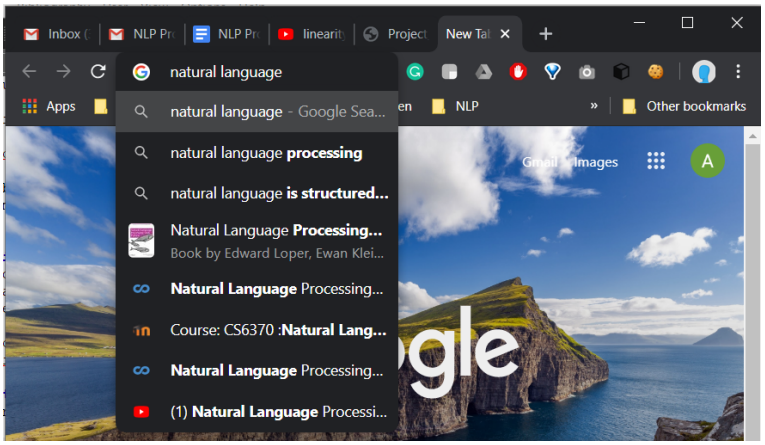

Fig. 5. query completion integrated with browser.

6.1 Query completion model

The details are given below:

1. A shallow neural network model was built, using an Embedding layer with 16 dimensions (each word), followed by an LSTM layer, to take context into account, followed by a Dense layer and an output layer, with softmax (to predict the next word).
2. All the sentences in the queries are preprocessed,like we take, say 5 continuous words, in all sentences and use then as train input sentences, and the next word as train output sentence.
   This can be illustrated by the example: "when constructing aeroelastic models of heated high speed aircraft". Here the set of train sentences constructed would be : ['when constructing aeroelastic models of', 'constructing aeroelastic models of heated', 'aeroelastic models of heated high',...], corresponding labels would be ['heated', 'high', 'speed',...].
3. Once the model is trained, we can pass an incomplete query and get the next 'n' words.

This is illustrated as shown in figure6 we predict next 2 words

```
In [53]:  1 complete_query("experimental studies of creep",2)
Out[53]: 'experimental studies of creep buckling either'
```

Fig. 6. query completion model illustration.

if we try to predict more words, the quality of words deteriorates, below, in figure7 we show, the result of predicting next 10 words.

We couldn't find a way, to compare with a base line model. Only manual interpretation of some queries were considered to evaluate this model.

```
In [14]:  1 complete_query("experimental studies of creep",10)
Out[14]: 'experimental studies of creep buckling or must be developed to the fundamental three speed'
```

Fig. 7. query completion model illustration with more next words.

References

1. Stemming Vs Lemmatization, which is better ?
2. Latent semantic analysis, wikipedia
3. KMeans clusering reference
4. Billel Aklouche1, Ibrahim Bounhas1, Yahya Slimani1, Query Expansion Based on NLP and Word Embeddings
5. https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa
6. https://monkeylearn.com/blog/introduction-to-topic-modeling/
7. David M. Blei, Andrew Y.Ng, Michael I. Jordan, Latent Dirchlet Allocation, Journal of Machine Learning Research 3 (2003) 993-1022
8. Centroid based classifier
9. LNCS Homepage, `http://www.springer.com/lncs`. Last accessed 4 Oct 2017