CS 6370 : Natural Language Processing
Final Project Report

Information Retrieval

by

Akash Reddy (EE17B001)
Nayan N (AE17B010)

Section 1 : Introduction

The objective of this project is to improve upon the information retrieval system that we had developed in the assignments during the course. There are broadly two Limitations with our earlier implementations that we hope to overcome. Detailed below are the Limitations, the Hypothesised solutions to counter these Limitations and the Observations and results obtained by these solutions. We implement these solutions as independent modifications to the baseline model to observe their effect in improving or deteriorating the model's performance.

Section 2 : Baseline

The baseline we use to judge whether the methodologies we implement are improvements is the information retrieval system that was developed over the two assignments. This model included using a tf-idf vector model to encode each document and query in the term space and cosine similarity with document query pairs were used to retrieve relevant documents. We use the nDCG and MAP scores obtained with this implementation as a baseline.

Section 3 : Limitation 1

Section 3.1 : Limitation Specification

The vanilla vector space model is constructed using the terms of all the documents in the corpus. In situations where the query contains synonymous words to those present in the document set, truly relevant documents may not be retrieved. In addition to synonymy of words, the vanilla model does not take word relatedness into consideration.

Example of situation: This may occur when the returned results of a query are all irrelevant, and their cosine similarities with the query are all low. This indicates a situation where the relevant documents are not highly cosine-similar to the query (therefore, their tf-idf vectors are not similar) and this could be due to the relevant documents containing related/synonymous words instead.

Section 3.2 : Hypothesised Solution

We hypothesise that extracting concepts/synsets and constructing the vector space model using these concepts would tackle the limitations detailed above. The expected effect is that related words are grouped under similar concepts thus improving the quality of retrieved documents.

Section 3.3 : Implementation

To build richer vector representations of documents from our dataset we employ the following techniques:

Section 3.3.1 : Latent Semantic Analysis (LSA):

LSA is a technique in NLP which is used to extract semantic relationships between terms of a document based on how frequently they co occur in all the documents from a given corpus. All terms in a corpus are condensed into a predetermined number of concepts or synsets using truncated Singular Value Decomposition (ref : Fig 3.3.1). This predetermined number is a hyperparameter which depends on the corpus we are working with. The term-document matrix which is represented by the tfidf vector representations is transformed into a concept-document matrix where each document is represented by a set of concepts. We transform the given query into the learnt concept space and find the cosine similarity between each document in our corpus. This allows us to rank the documents based on their perceived relevance to a given query. [1]

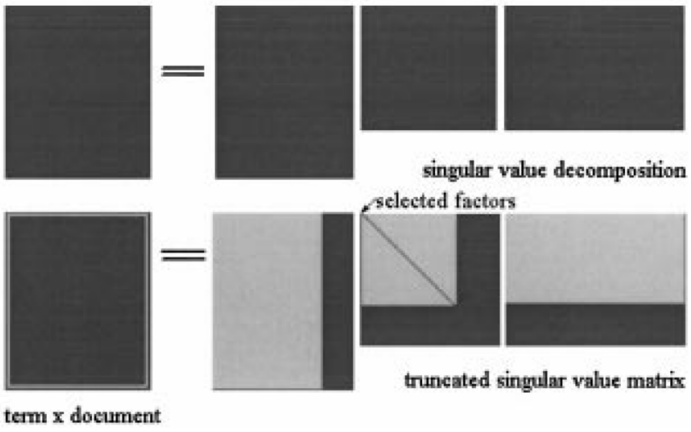

Fig 3.3.1.1 : Truncated SVD src

Section 3.3.2 : Latent Dirichlet Allocation (LDA):

LDA is broadly a method of topic modelling. It is defined as a generative statistical model that allows sets of observations (in this case documents) to be explained by unobserved (learnt from data) groups (topics/concepts) that quantify the similarity between the sets of observations. LDA is an advancement over LSA in two stages:

1. It uses a probabilistic method instead of SVD to quantify semantic relations between terms which is learnt by modelling the probabilities of observing a topic given a document, a term given a topic and the prior probability of a document itself. These probabilities are used to build a word-document representation. This method is often called probabilistic LSA or pLSA. This can be condensed into one formula shown below which can draw parallels to the SVD formula seen in the previous section. Here Z is the topic, P(W|Z) and P(D|Z) is the probability of observing a word and a document respectively given a topic.

$$P(D,W) = \sum_{Z} P(Z)P(D|Z)P(W|Z)$$

$$A \approx U_t S_t V_t^T$$

Fig 3.3.2.1 : pLSA Formulation

2. LDA is the bayesian version of pLSA where dirichlet priors are used for the document-topic and word-topic distributions. This often leads to better generalization. The pipeline of LDA is as follows :

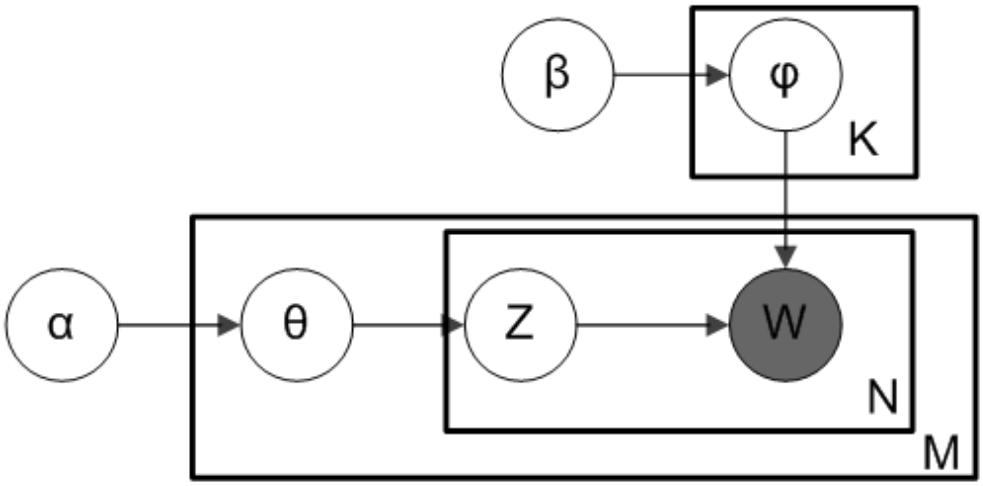

Fig 3.3.2.2 : LDA Model

Here, from the dirichlet distribution Dir(α), we draw a random sample representing the topic distribution 'θ' from which we derive the topic Z. From another dirichlet distribution Dir(β) we select a random sample representing the word distribution of the given topic Z. This word distribution is called 'ϕ' from which we choose a word W.

LDA allows us to extract human interpretable topics which is helpful in understanding where our model is performing better and where it is otherwise lacking. [2] [3]

Section 3.3.3 : Word2Vec (W2V):

Word2Vec is a technique in NLP which uses a neural network to learn word associations from a large corpus of text and the advantage is that these weights are often learnt using large datasets and can be directly used out of the box to get word embeddings. These models are trained using a Continuous Bag of Words approach where given a coset of context/surrounding words they are expected to predict the missing word or skip gram where the model is expected to predict the surrounding/context words given the word at a certain position. It thus learns an aspect of word relatedness which is used to encode meaning into the vector representation. The hyperparameter to be tuned is the number of features that each word is to be encoded into. We then average out all the encoded features to obtain a document encoding. [4] [5]

Section 3.4 : Observations

As we can see from the model details above, all 3 of them have their own salient features and it is quite unclear from direct inspection which of them would yield the best results. We noticed that the size and

type of our dataset played a huge role in determining how effective the methods were in improving the results of our information retrieval system. It is important to note that our dataset is rather small with only about 1400 entries and it is very specific in its content. This makes it hard to implement the word2vec model as it may not include the best of domain knowledge as well as LDA as the package we used has the drawback of suffering from bad topic representation when large numbers of topics are extracted from a small dataset. Additionally since all of these techniques effectively work as dimension reduction techniques they drastically reduce the runtime of the information retrieval script. Here are some detailed observations for all three methodologies :

Model	Observations
LSA	1. This was the simplest one to implement and ended up being the most effective. We had to only tune the ideal number of components to be extracted which was found to be 300 components or topics. 2. The drawback is that these extracted documents are not human interpretable and thus cannot be used to study these aspects of the dataset.
LDA	1. This model was much harder to implement and tune as there are quite a few hyperparameters which require proper tuning. 2. It has the added advantage that the topics extracted can be mapped back to words hence it gives a better idea about the dataset. 3. We were unable to learn effective topic distributions for the given dataset as the model would not learn good topic representations above around 50 topics but these topics were not enough to correctly represent a document and a query.
W2V	1. This method was rather easy to implement in that it requires no training and directly gives us word embeddings which are then averaged out to obtain sentence embeddings. The hyperparameter which represents the size of the word embedding to be extracted does not play a big role in the downstream task. 2. The issue here is that the word embeddings probably do not encode enough dataset knowledge as they are trained on general corpora and not on our model. Additionally when the word embeddings are averaged, it is a possibility that some features are lost while averaging which leads to poorer results.

Table 3.4.1 : Observations of different Models

Section 3.4 : Results

Listed below are the results for the best performing model for each of the three implementations. Note that as stated above we compare MAP and nDCG metrics to evaluate our models :

Model	Final Hparam	nDCG (avg increase)	MAP (avg increase)
LSA	n_components=300	2.5%	3.0%
LDA	num_topics = 500	-35%	-30%
W2V	emb_size = 75	-30%	-35%

Table 3.4.1 : Results and comparative improvements

Hence, LSA with the tf-idf vector model performed the best and achieved a 2.5% increase in the nDCG scores and 3.0% increase in the MAP scores over the baseline model. Comprehensive results can be found in the output folder of the code base.

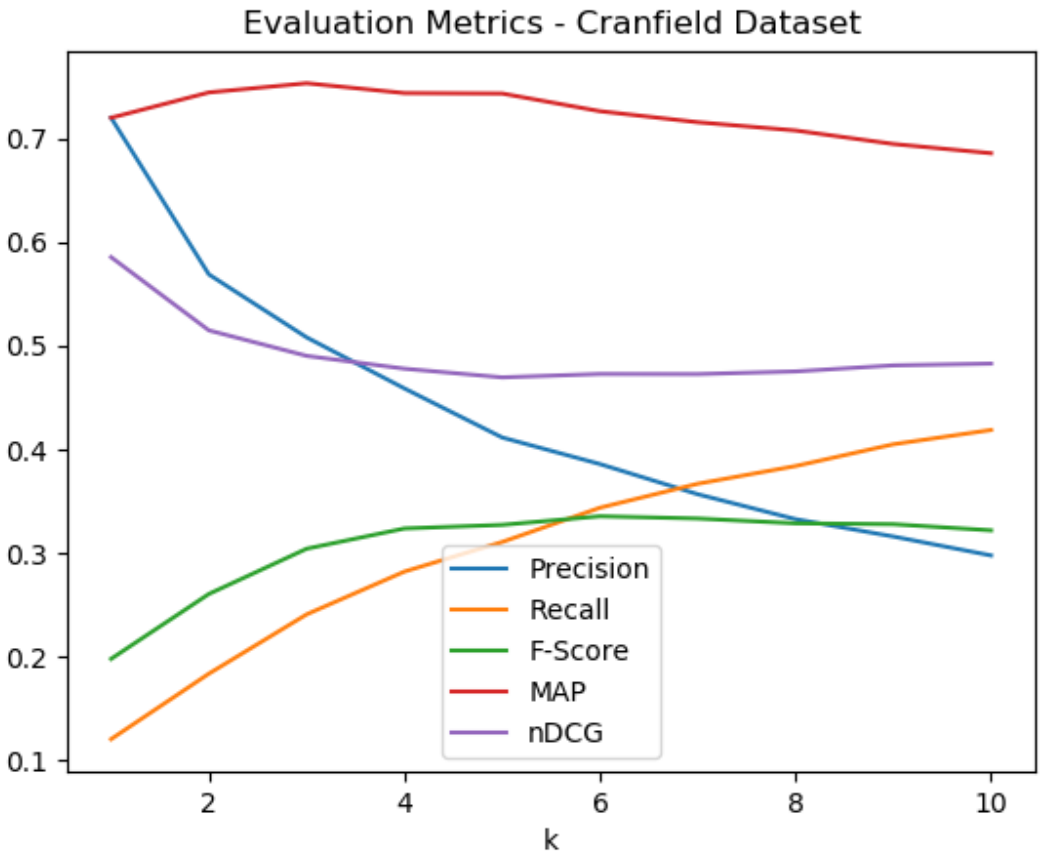

Figure 3.4.1 : Metric graph for LSA with 300 components

Model	**k**	**MAP**	**nDCG**
Baseline	1	0.65	0.53
	5	0.69	0.45
	10	0.66	0.46
LSA (300)	1	0.72	0.59
	5	0.73	0.47
	10	0.69	0.48

Table 3.4.2 : Metrics for best model vs baseline model

Section 4 : Limitation 2

Section 4.1 : Limitation Specification

In some cases, a single term may have multiple meanings (polysemy). In these situations, the vanilla vector space model based information retrieval system would return poorer results as it may return documents in which a different sense of the word is used.

Example of situation: This may be occurring in the opposite situation as in L1 - when the returned results of a query are all irrelevant, but their cosine similarities are all high. This means that, despite sharing common words, the document is not relevant. For example, in the Cranfield dataset, the word "number" can mean different things in "Reynold's number" and "number of ways".
This could sometimes be resolved by ensuring that the right sense of the polysemous words is included, so that documents with the wrong sense of the word are not ranked higher up.

Section 4.2 : Hypothesised Solution

We hypothesise that using the assistance of Word Sense Disambiguation, we can reduce the influence of polysemy on the ranking of the IR system.

Section 4.3 : Method and Implementation

The approach we consider towards WSD is an unsupervised WSD algorithm based on the method highlighted in the book titled "Statistical Language Learning" by Eugene Charniak (pg. 151-155). The method is originally authored by H. Schutze in the publications titled "Word Sense Disambiguation with Sublexical Representations" and "Context Space".

4.3.1 : Unsupervised Context-Based Word Sense Disambiguation (WSD):

The method of unsupervised WSD used is based on second-order context vectors and their subsequent hierarchical clustering. To obtain a good idea of word senses, not just the neighbouring words are used as context, but also the words that they in turn usually occur in the neighbourhood of.

For each word *type* w^i, we define an associated vector $A(w^i)$ as:

$$A(w^i) = \sum_{k=1}^{n} \delta(i,k) \; < c_k^{\,1}, c_k^{\,2}, c_k^{\,3}, \ldots, c_k^{\,W} >$$

Here n is the number of *tokens* in the corpus, W is the number of *types* in the corpus, and $c_k^{\,t}$ is the number of times type w^t appears in a 100-token window (±50 tokens) around token w_k, excluding token w_k itself.

$\delta(i,k)$ is an indicator function of whether the token w_k is of type w^i. Therefore,

$\delta(i,k) = 1\ if\ w_k = w^i,\ else\ 0.$

Therefore, $A(w^i)$ captures the average first-order context of type w^i. In order to capture the second-order context, the true context vector for a token is actually defined as:

$$C(w_j) = \sum_{i=1}^{W} c_j^{\,i} A(w^i)$$

Through this formulation, the context vector associated with a token w_j becomes a weighted average of the first-order context vectors of the types surrounding the token itself. Therefore $C(w_j)$ is a second-order context vector for token w_j.

Finally, the similarity measure used between vectors is the cosine similarity, as the vectors may be of unequal Euclidean length, and the angular similarity/distance between them in the word-type vector-space is what is important.

The formation of clusters and classification of contexts in Schutze's method uses AutoClass and Buckshot (two clustering methods from other references - by Cheesman et al. and Cutting et al. respectively).

4.3.2 : Implementation Details, Modifications and Assumptions:

- In our implementation, since we have documents in the corpus and we can assume that each document is about one topic, we can do a more topic-based context modelling. Therefore, the context window for each token has been defined as all the tokens for the document in which it is present, as opposed to the above defined 100-token context window. Accordingly, $c_k^{\,i}$ also changes to account for the counts in the document.
- This assumption allows us to run a large number of operations using optimised matrix operations on Python. For example,
    - $c_k^{\,i}$ and the corresponding types w^i can be calculated using collections.Counter package..
    - The vector $< c_k^{\,1},\ c_k^{\,2},\ c_k^{\,3},\ ....,\ c_k^{\,W} >$ can therefore be calculated using these token counts in the particular document that token w_k is within. The only modification is that the count of type wk corresponding to token w_k reduces by 1 (since the token w_k itself is to be excluded in the context).
    - $$A(w^i) = \sum_{k=1}^{n} \delta(i,k)\ < c_k^{\,1},\ c_k^{\,2},\ c_k^{\,3},\ ....,\ c_k^{\,W} >$$
      $$= \sum_{k=1}^{d_1} \delta(i,k)\ < c_k^{\,1},\ c_k^{\,2},\ c_k^{\,3},\ ....,\ c_k^{\,W} > + \sum_{k=d_1+1}^{d_2} \delta(i,k)\ < c_k^{\,1},\ c_k^{\,2},\ c_k^{\,3},\ ....,\ c_k^{\,W} > +$$

$$\ldots + \sum_{k=d_{M-1}+1}^{d_M} \delta(i,k) \; < c_k^{\,1},\, c_k^{\,2},\, c_k^{\,3},\, \ldots.,\, c_k^{\,W} >$$

$$= \sum_{k=1}^{d1} \delta(i,k) \; < 1_k^{\,doc1}, 2_k^{\,doc1}, \ldots.,\, W_k^{\,doc1} > +$$
$$\sum_{k=d1}^{d2} \delta(i,k) \; < 1_k^{\,doc2}, 2_k^{\,doc2}, \ldots.,\, W_k^{\,doc2} > + \ldots +$$
$$\sum_{k=d_{M-1}}^{d_M} \delta(i,k) \; < 1_k^{\,docM}, 2_k^{\,docM}, 3_k^{\,docM}, \ldots.,\, W_k^{\,docM} >$$

$$= < 1_k^{\,doc1}, 2_k^{\,doc1}, \ldots.,\, W_k^{\,doc1} > (\sum_{k=1}^{d1} \delta(i,k)) +$$
$$< 1_k^{\,doc2}, 2_k^{\,doc2}, \ldots.,\, W_k^{\,doc2} > (\sum_{k=d1}^{d2} \delta(i,k)) + \ldots +$$
$$< 1_k^{\,docM}, 2_k^{\,docM}, 3_k^{\,docM}, \ldots.,\, W_k^{\,docM} > (\sum_{k=d_{M-1}}^{d_M} \delta(i,k)) \quad (\#)$$

Here, d_m represents the last token in document m, assuming the tokens are arranged in the order in which they appear in document 1 to document m.

In each summation term's vector, $W_k^{\,docm}$ represents the number of times type W appears in document m (this quantity can be used in place of $c_k^{\,W}$ as the context of token w_k has been defined as the entire document in which w_k is present.

If wk represents the *type* of token w_k, then $wk_k^{\,docm}$ = (number of times type wk appears in the document m) - 1. This is the only modification to be made in the above vector, because the token w_k itself has to be excluded from the context.

- Considering all this, it is possible to define each summation term in (#) as a scala-vector multiplication. When we traverse through a particular document m, let
$\delta = \; < \sum_{k=d_{m-1}}^{d_m} \delta(1,k), \sum_{k=d_{m-1}}^{d_m} \delta(2,k), \sum_{k=d_{m-1}}^{d_m} \delta(3,k), \ldots., \sum_{k=d_{m-1}}^{d_m} \delta(W,k) >$ for all types. We calculate the association vector $A(w^i)$ by accumulating $< 1_k^{\,docM}, 2_k^{\,docM}, 3_k^{\,docM}, \ldots.,\, W_k^{\,docM} > (\sum_{k=d_{m-1}}^{d_m} \delta(i,k))$ for all documents m. We can

store all $A(w^i)$ vectors in a matrix A, stacked row-wise.

- When we do this, we also have the c_j^i values (or counts of every type w^i in the context of token w_j) because we have calculated them already. If we wish to calculate the final token-context-vector for token w_j, we arrange the row vector $< c_j^1, c_j^2, ..., c_j^W >$ and find its vector-matrix multiplication with the matrix of association vectors A to get the context vector of token w_j. This is equivalent to calculating $C(w_j) = \sum_{i=1}^{W} c_j^i A(w^i)$ as given in the above formulation of the solution.

- Any token that has less than 10 incidences in the entire corpus is not considered for disambiguation, as their disambiguation will not amount to much. Similarly, any token that has more than 10 incidences, but only across less than 5 documents are also ignored (since each document is one context). We can assume that not many documents will be missed or ranked low, if we fail to perform clustering of just 5 contexts of a certain word type. This also allows us to save memory and compute, by skipping the insignificant types during disambiguation.

  On the other hand, some tokens have >100 incidences across >50 documents. For them, disambiguation may be useful.

- After forming all second-order token context vectors $C(w_j)$, instead of AutoClass or Buckshot, we have performed one hierarchical clustering each, on all the token vectors of each type. Hierarchical clustering helps us tune the degree of clustering by changing the height of the top-down dendrogram at which we want to cluster the vectors (clustering at the highest point of the top-down dendrogram gives us only 1 cluster for all tokens of the type, while clustering at the bottom-most point gives us 1 cluster for each token).

  As a result, we can try to obtain the best possible clustering of senses of a word type (or disambiguation).

- We do a similar process with the queries. After we map any token (say “experiment”) into this context vector-space and we obtain the cluster (say cluster 1) that it is closest to by cosine similarity, we do not replace the original “experiment” by “experiment1”. Instead, we add a token “experiment1” after the token “experiment”. This is so that the connection between different senses of “experiment” is still preferred. Especially in the case of fine sense disambiguation, the different senses of the word “experiment” would still be close to each other.

  If we replace the token “experiment” completely with “experiment1”, and another with “experiment2”, then the two tokens which were originally the same token become completely unrelated. This is undesirable. By simply **adding** “experiment1” or “experiment2” after the

original token, the connection between the two documents or queries is preserved.

- Finally, the way we use WSD is as follows: We obtain the relevant documents in order without using WSD for each query. Then, we take the top 10 documents of this order for the query, and sort them after taking the disambiguated versions of the documents and the query and using cosine similarity. This way, we hope to obtain a better ranking and hence an improved MAP@10 and/or nDCG@10 on the IR system.

Section 4.4 : Observations, Results and Learnings

Despite the simplifying assumptions made according to Section 4.3.2, it turned out that a large amount of memory was required to run it on 1400 documents. The WSD takes ~14 minutes to run on an 8GB RAM 2.7 GHz, Intel Core i7 7th Gen Machine, with all programs closed. Therefore, the memory usage is high (storage of the association vectors matrix with no. of elements of the order 10^8). Similarly, storage and returning of the word context vectors (>100000 vectors each of dimension 8455) are very demanding.

On the other hand, the time taken if we take just the first 500/1400 documents of the Cranfield dataset was only ~30 seconds. Therefore, scalability proves to be an issue.

By choosing a mild form of clustering (clustering height = 95% of maximum dendrogram height), we were able to obtain the following results:

Model	**k**	**MAP**	**nDCG**
Baseline	1	0.653	0.528
	5	0.702	0.457
	10	0.658	0.467
Unsupervised WSD	1	0.604	0.492
	5	0.680	0.444
	10	0.639	0.458

Table 3.4.2 : Metrics for best model vs baseline model

We observe a slight decline in performance. We conjecture that this is due to one or more of the following potential drawbacks:

- While we had almost implemented WSD, we realised that WSD may not improve the performance of an IR system by much at all. We hypothesise that this is due to the fact that, usually, query tokens also have neighbouring words that are present in the document we are

looking for. Taking the “number” example, a query looking for “Reynolds number” documents may also have “Reynolds number” in it already, instead of just “number”. However, through the second-order context information, we hoped to see some small performance improvements at least.

- Once the similarity measures between different contexts of a word are established, they are hard-clustered into one sense. They lose their strong sense of connection with other senses of the same word. A more soft-clustered/similarity/relatedness based approach may have done better.
- Our assumption of ignoring all tokens with no more than 10 incidences, or spread across no more than 5 documents may be incorrect. Also, our assumption that the context window can be the entire document may also be invalid.
- Our implementation of unsupervised WSD may have some errors.

****