Literature Survey (Background)

<Insert brief explanation of important concepts like Cosine-Similarity, TF-IDF>

The problem of keyword recommendation is challenging due to the complexity of English language and the difficulty of evaluating if a word or phrase is important to the document. Keyword recommendation algorithms either use a corpus in addition to the query document or perform cooccurence analysis on the query document alone.

Single Document Approaches:

In the single document approaches, the algorithms use only the query document to suggest the keywords.

Matsuo and Ishizuka [reference from Lott] implement a method which constructs the word co-occurence matrix 'M' to help in keyword suggestion.

<Insert matrix image>

Here, each entry in the matrix 'M[i][j]' corresponds to the number of times the two words 'w_i' and 'w_j' occur in the same sentence in the document. Matsuo and Ishizuka claim that important words cooccur with other words more often than if each occurrence of the word was randomly distributed in the document. With this assumption, important words in the document have a co-occurence vector with a large deviation from the expected vector if the word was randomly distributed. For this, the authors perform Pearson's chi-squared test (X^2). Words with a higher chi-squared value have are considered as keywords.

In addition to methods which utilize frequency analysis alone, some methods use clustering of related words/phrases into groups to determine the keywords for the document. Ohsawa et al [reference from Lott] implement a method named KeyGraph which constructs a network with the terms in the document as nodes, and the frequent co-occurences between terms as the edges.

<Insert network image>

After the graph is constructed, clusters in the graph are obtained by finding the maximally connected subgraphs. Candidate keywords are selected by considering nodes (terms) that have edges between two distinct clusters. The candidate keywords are then sorted by the probability of the nodes being used in joining the clusters. The top-ranked nodes (terms) are suggested as keywords for the document.

<Insert content about PageRank method>

Corpus Based Approaches

1) TFIDF Approach

Early work in the field mostly involved analysis of the term frequency in a single document for keyword extraction. The Term Frequency – Inverse Document Frequency approach takes into account a corpus to help score the terms in a document while also considering the other documents in the corpus. The TFIDF statistic is computed for each term in each document using the formula:

<Insert TFIDF formula here>

This statistic gives higher score to terms which are more frequent in the document, while at the same time reducing the score for words which are very common across the whole corpus. This makes sure that words which are important and unique to the document are given a high score. To find the keyword suggestions for a document, the terms in the document are ranked based on their TF-IDF scores, and the top terms in this ranking are suggested as keywords.

2) Vector Representation Approaches

While the above method directly uses the TFIDF metric to find the keywords in a document, some methods train models using a corpus, where each document in the corpus is represented using a vector. After the feature representation is performed these models take advantage of Vector Similarity or Machine Learning tools such as classification to suggest keywords for new documents.

a) Classification Approach

In this approach [reference : poster and others], the problem of suggesting keywords is modified into a multilabel-binary classification problem. Text documents in the corpus can be represented as fixed size feature vectors using multiple methods such as TFIDF, LDA and LSA. These different methods can be used individually or in conjunction with each other to represent the documents. Once the documents are represented as feature vectors, a binary vector 'L_d' is constructed for each document 'd' to represent its keywords. Considering the list of all possible keywords in the corpus as 'K' :

<Equation form>
L_d_i = 1 if K_i is a keyword is a keyword for the document 'd', else L_d_i = 0.

This binary vector L_d acts as the label vector for the document 'd'. After the dataset is prepared, a Multi-Label classifier is trained using this dataset. In the prediction step, a new document 'q' is converted into the feature vector representation using TFIDF, LDA, LSA or a combination of these approaches. This conversion method should be the same as used in training. Once the query document is represented as a feature vector, the trained classifier is used to find the keywords.

b) Document Similarity

Tuarob et al, also use a corpus and a vector representation of documents to help in keyword extraction. The main idea of this paper is to use the similarity of the query document to each of the training documents in order to suggest keywords from a tag library. Three major concepts are involved in this technique namely: Term Frequency – Inverse Document Frequency (TF-IDF), Latent Dirichlet Allocation (LDA) and Cosine-Similarity. <Brief explanation of these will be given in background section>.

Given a document corpus D = <d1, d2, .. dn>, a keyword library T = <t1, t2, … tn>, a query document 'q', the algorithm outputs T* = <t1, t2, .. tk> which is a ranked list of keywords and is a subset of the keyword library T.

The major steps involved in the algorithm are:

Step 1: Initially all the documents in the corpus are converted into a vector format using either TF-IDF or LDA.
Step 2: P(t | q, T, D, M) is computed for each tag in the corpus using:
<Insert P() equation here>
<Insert TagScore equation here>
Here, M is the document similarity measure. It can be the cosine similarity between either the TF-IDF or LDA vector representations of query document and the documents in the corpus.
Step 3: All keywords in the Keyword Library T, are ranked based on their P(t | q, T, D, M)

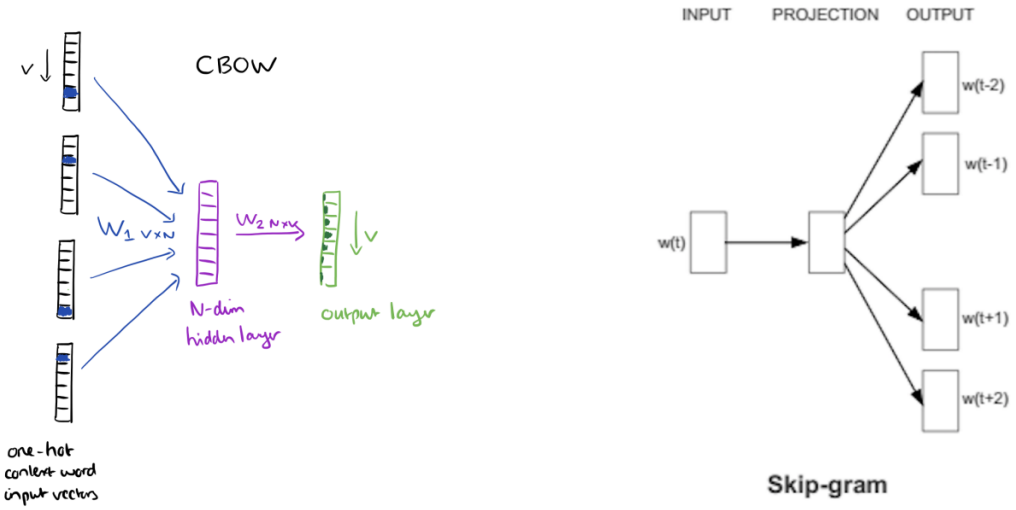
With extensive analysis on 4 datasets (DAAC, DRYAD, KNB and TREEBASE), the authors show that the document similarity approach using both TF-IDF and LDA (Topic Modeling) outperform the baseline KEA algorithm. The evaluation of the algorithms has been performed using metrics such as Precision, Recall, F-Score which consider the suggested keywords as an unordered set, and Mean Reciprocal Rank (MRR) and Binary Preference (Bpref) which take into consideration the ranks of the suggested keywords.

Paragraph Vectors Approach

Due to further advances in Natural Language Processing with the help of Neural Networks, additional methods to represent text such as Word2Vec and Paragraph Vectors have been created. These representations perform better than previous techniques such as Bag of Words (BOW) and Term Frequency – Inverse Document Frequence (TF-IDF) at many semantic relatedness tasks.

We use the Paragraph Vectors approach for document embedding in order to obtain keywords. The Paragraph Vectors algorithm is inspired by Word2Vec.

Word2Vec



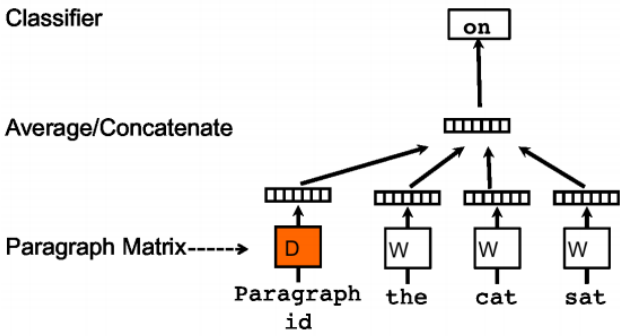
Word2Vec is a word embedding technique, where the words from a large corpus are represented as feature vectors. This utilizes Neural Networks with a single hidden layer. The task of the network is to predict a word, given the context of the word.

The input layer of this neural network is the One-Hot encoding of the context words (the words that surround the current word), and the output layer is the vector for the word under consideration. The weights of the network are initialized randomly and the network is trained using Stochastic Gradient Descent. The value of the gradient is obtained using the backpropogation algorithm. During training, the algorithm parses through a large corpus of text, and for each occurrence of a word, the weights of the neural network are updated using Gradient Descent. After multiple iterations through the corpus, the training converges, and the hidden layers after convergence represent the feature vectors for the words in the output layer.

<Insert training equations (Gradient Descent)>

The advantage of this technique compared to BoW is that, similar words have feature vectors which are closer to each other in the vector space. This technique can also be used for answering analogy questions, which demonstrates the ability of the feature vectors in capturing the semantics of the words.

Paragraph Vectors



The idea of learning paragraph vectors was inspired by the method used for training word vectors. Similar to the word vector training, the task of the network is to predict the next word given the contexts from the paragraph it occurs in. In addition to the word vectors (W), the network also takes as input a unique vector for each paragraph (D).

<Insert new equation for hidden layer>

This paragraph vector is constant for all contexts sampled from the same paragraph and the word vectors are constant across all paragraphs. The Paragraph Vector can be seen as containing the missing information from the current context which is required to predict the next word. Due to this, the model is also called Distributed Memory Model of Paragraph Vectors (PV-DM).

Similar to word vector training, this network is trained using Stochastic Gradient Descent and the gradient at each step is obtained using the backpropogation algorithm. At each step of training, a fixed length (of words) context is sampled from a random parapgraph, the paragraph token and word vectors corresponding to the context are taken as input. The error and error gradient are computed from the network, which are used to update the weights of the model.

Once the training converges, the unique vectors for each paragraph are treated as the feature representations of the paragraphs observed in training. To obtain the feature representation for a new (unseen) paragraph, an inference step is performed, where the parameters of the network and the word vectors are kept constant. Additional columns are added to the matrix D (paragraph vector matrix) and gradient descent is performed on D. With this inference step we obtain a feature representation for the new paragraphs. This model can be extended from using paragraphs to documents, where we use a unique vector for each document instead of each paragraph. This results in a feature representation for the individual documents instead of the paragraphs in the document.

The advantage of the Paragraph Vectors approach for representing documents compared to BoW or TF-IDF approaches is that, these Vectors represent the semantic information contained in a paragraph instead of just the counts of words. This helps represent the documents in a better way for NLP tasks such as calculating document similarity, which is used for our keyword extraction. Also, the word order is preserved, in each context without actually taking into consideration, all the n-grams in the corpus. This helps keep the dimensionality of the feature vectors small compared to n-gram models.

Keyword Extraction

Once all the documents in the corpus are represented as feature vectors, keyword extraction is done using document similarity. Consider the document corpus D, and the list of all tags in the corpus K, a query document q. Let F, be the Paragraph Vector representations of all the documents from D obtained after training the network.

Algorithm to obtain the keyword suggestions for the query document q: <Use Latex algorithm notation>
<Use equations for the steps>

Step 1) To obtain the keyword suggestions for the query documnt q, we first perform the inference step on the network and obtain the paragraph vector representation for q.

Step 2) We iterate through all the documents in the corpus, and find the similarity of the query document's vector with the parapgraph vector of each document in the corpus.

Step 3) For each document 'd' in the corpus, the similarity score with the query vector doc-sim(q,d) is added to the list Lk of similarities for all the keywords for the corresponding document.

Step 4) After we iterate through the complete corpus, we have a list of similarity scores for each keyword 'k' in the keyword list K. For each keyword 'k' we consider the maximum (/sum/avg) of all the entries in the list 'Lk' of similarities for 'k'.

<Insert equations here>

Step 5) To finally obtain the keyword suggestions, we consider the top keywords, based on the Similarity score from step 4.