Department of Computer Engineering
Academic Term: Jan-April 2022
Rubrics for Mini Project

Class : B.*E. Computer* **Subject Name :*NLP***
Semester : VIII **Subject Code :*DLO8012***

Practical No:	**Mini project**
Title:	**Keyword Extraction for IR systems**
Date of Performance:	**11/05/2022**
Roll No:	**8626**
Name of the Student:	**Divita Phadakale**

Rubric for Mini Project

Indicator	Very Poor	Poor	Average	Good	Excellent
Timeline: **Maintains project deadline (2)**	Project not done (0)	More than two session late (0.5)	Two sessions late (1)	One session late (1.5)	Early or on time (2)
Completeness: **Complete all parts of project (2)**	N/A	< 40% complete (0.5)	~ 60% complete (1)	~ 80% complete(1.5)	100% complete(2)
Application design: **(4)**	Design aspects are not used (0)	Poorly designed (1)	Project with limited functionalities (2)	Working project with good design (3)	Working project with good design and advanced techniques are used (4)
Presentation(2)	Not submitted report (0)	Poorly written and poorly kept report(0.5)	Report with major mistakes(1)	Report with less than 3-4 mistakes (1.5)	Well written accurate report(2)

Total marks: **Signature of Teacher:**

Table of Contents

Introduction

What is Keyword Extraction

Keyword extraction (also known as keyword detection or keyword analysis) is a text analysis technique that automatically extracts the most used and most important words and expressions from a text. It helps summarize the content of texts and recognize the main topics discussed.

Keyword extraction uses machine learning artificial intelligence (AI) with natural language processing (NLP) to break down human language so that it can be understood and analyzed by machines. It's used to find keywords from all manner of text: regular documents and business reports, social media comments, online forums and reviews, news reports, and more.

Imagine you want to analyze thousands of online reviews about your product. Keyword extraction helps you sift through the whole set of data and obtain the words that best describe each review in just seconds. That way, you can easily and automatically see what your customers are mentioning most often, saving your teams hours upon hours of manual processing.

Why is Keyword Extraction important?

With keyword extraction you can find the most important words and phrases in massive datasets in just seconds. And these words and phrases can provide valuable insights into topics your customers are talking about.

Considering that more than 80% of the data we generate every day is unstructured — meaning it's not organized in a predefined way, making it extremely difficult to analyze and process – businesses need automated keyword extraction to help them process and analyze customer data in a more efficient manner.

What percentage of customer reviews are saying something related to Price? How many of them are talking about UX? These insights can help you shape a data-driven business strategy by identifying what customers consider important, the aspects of your product that need to be improved, and what customers are saying about your competition, among others.

In the academic world, keyword extraction may be the key to finding relevant keywords within massive sets of data (like new articles, papers, or journals) without having to actually read the entire content.

Whatever your field of business, keyword extraction tools are the key to help you automatically index data, summarize a text, or generate tag clouds with the most representative keywords. Some of the major advantages of keyword extraction include:

Scalability

Automated keyword extraction allows you to analyze as much data as you want. Yes, you could read texts and identify key terms manually, but it would be extremely time-consuming. Automating this task gives you the freedom to concentrate on other parts of your job.

Consistent criteria

Keyword extraction acts based on rules and predefined parameters. You don't have to deal with inconsistencies, which are common in manual text analysis.

Real-time analysis

You can perform keyword extraction on social media posts, customer reviews, surveys, or customer support tickets in real-time, and get insights about what's being said about your product as they happen and follow them over time.

How does Keyword Extraction Work?

Keyword extraction simplifies the task of finding relevant words and phrases within unstructured text. This includes emails, social media posts, chat conversations, and any other types of data that are not organized in any predefined way.

Keyword extraction can automate workflows, like tagging incoming survey responses or responding to urgent customer queries, allowing you to save huge amounts of time. It also provides actionable, data-driven insights to help make better business decisions. But the best thing about keyword extraction models is that they are easy to set up and implement.

There are different techniques you can use for automated keyword extraction. From simple statistical approaches that detect keywords by counting word frequency, to more advanced machine learning approaches that create even more complex models by learning from previous examples.

In this section, we'll review the different approaches to keyword extraction, with a focus on machine learning-based models.

Simple Statistical Approaches

Using statistics is one of the simplest methods for identifying the main keywords and key phrases within a text.

There are different types of statistical approaches, including word frequency, word collocations and co-occurrences, TF-IDF (short for term frequency–inverse document frequency), and RAKE (Rapid Automatic Keyword Extraction).

These approaches don't require training data to extract the most important keywords in a text. However, because they only rely on statistics, they may overlook relevant words or phrases

that are mentioned once but should still be considered relevant. Let's look at some of these approaches in detail:

Word Frequency

Word frequency consists of listing the words and phrases that repeat the most within a text. This can be useful for a myriad of purposes, from identifying recurrent terms in a set of product reviews, to finding out what are the most common issues in customer support interactions.

However, word frequency approaches consider documents as a mere 'bag of words', leaving aside crucial aspects related to the meaning, structure, grammar, and sequence of words. Synonyms, for example, can't be detected by this keyword extraction method, dismissing very valuable information.

Word Collocations and Co-occurrences

Also known as N-gram statistics, word collocations and co-occurrences help understand the semantic structure of a text and count separate words as one.

Collocations are words that frequently go together. The most common types of collocations are bi-grams (two terms that appear adjacently, like 'customer service', 'video calls' or 'email notification') and trigrams (a group of three words, like 'easy to use' or 'social media channels').

Co-occurrences, on the other hand, refer to words that tend to co-occur in the same corpus. They don't necessarily have to be adjacent, but they do have a semantic proximity.

TF-IDF

TF-IDF stands for term frequency–inverse document frequency, a formula that measures how important a word is to a document in a collection of documents.

This metric calculates the number of times a word appears in a text (term frequency) and compares it with the inverse document frequency (how rare or common that word is in the entire data set).

Multiplying these two quantities provides the TF-IDF score of a word in a document. The higher the score is, the more relevant the word is to the document.

TD-IDF algorithms have several applications in machine learning. In fact, search engines use variations of TF-IDF algorithms to rank articles based on their relevance to a certain search query.

When it comes to keyword extraction, this metric can help you identify the most relevant words in a document (the ones with the higher scores) and consider them as keywords. This can be particularly useful for tasks like tagging customer support tickets or analyzing customer feedback.

In many of these cases, the words that appear more frequently in a group of documents are not necessarily the most relevant. Likewise, a word that appears in a single text but doesn't appear in the remaining documents may be very important to understand the content of that text.

Let's say you are analyzing a data set of Slack reviews:

Words like this, if, the, this or what, will probably be among the most frequent. Then, there will be a lot of content-related words with high levels of frequency, like communication, team, message or product. However, those words won't provide much detail about the content of each review.

Thanks to the TF-IDF algorithm, you are able to weigh the importance of each term and extract the keywords that best summarize each review. In Slack's case, they may extract more specific words like multichannel, user interface, or mobile app.

RAKE

Rapid Automatic Keyword Extraction (RAKE) is a well-known keyword extraction method which uses a list of stopwords and phrase delimiters to detect the most relevant words or phrases in a piece of text.

Take the following text as an example:

Keyword extraction is not that difficult after all. There are many libraries that can help you with keyword extraction. Rapid automatic keyword extraction is one of those.

The first thing this method does is split the text into a list of words and remove stopwords from that list. This returns a list of what is known as content words.

Suppose our list of stopwords and phrase delimiters look like these:

stopwords = [is, not, that, there, are, can, you, with, of, those, after, all, one] delimiters = [., ,]

Then, our list of 8 content words, will look like this:

content_words = [keyword, extraction, difficult, many, libraries, help, rapid, automatic]

Then, the algorithm splits the text at phrase delimiters and stopwords to create candidate expressions. So, the candidate keyphrases would be the following:

Keyword extraction is not that difficult after all. There are many libraries that can help you with keyword extraction. Rapid automatic keyword extraction is one of those.

Once the text has been split, the algorithm creates a matrix of word co-occurrences. Each row shows the number of times that a given content word co-occurs with every other content word in the candidate phrases. For the example above, the matrix looks like this:

	keyword	extraction	difficult	many	libraries	help	rapid	automatic
keyword	3	3	0	0	0	0	1	1
extraction	3	3	0	0	0	0	1	1
difficult	0	0	1	0	0	0	0	0
many	0	0	0	1	1	0	0	0
libraries	0	0	0	1	1	0	0	0
help	0	0	0	0	0	1	0	0
rapid	1	1	0	0	0	0	1	1
automatic	1	1	0	0	0	0	1	1

After that matrix is built, words are given a score. That score can be calculated as the degree of a word in the matrix (i.e. the sum of the number of co-occurrences the word has with any other content word in the text), as the word frequency (i.e. the number of times the word appears in the text), or as the degree of the word divided by its frequency.

If we were to compute the degree score divided by the frequency score for each of the words in our example, they would look like this:

Word	Degree Score
keyword	2.66
extraction	2.66
difficult	1.0
many	2.0
libraries	2.0
help	1.0
rapid	4.0
automatic	4.0

Those expressions are also given a score, which is computed as the sum of the individual scores of words. If we were to calculate the score of the phrases in bold above, they would look like this:

keyword extraction	5.33
many libraries	4.0
rapid automatic keyword extraction	13.33

If two keywords or keyphrases appear together in the same order more than twice, a new keyphrase is created regardless of how many stopwords the keyphrase contains in the original text. The score of that keyphrase is computed just like the one for a single keyphrase.

A keyword or keyphrase is chosen if its score belongs to the top T scores where T is the number of keywords you want to extract. According to the original paper, T defaults to one third of the content words in the document.

For the example above, the method would have returned the top 3 keywords, which, according to the score we have defined, would have been rapid automatic keyword extraction (13.33), keyword extraction (5.33), and many libraries (4.0).

Linguistic Approaches

Keyword extraction methods often make use of linguistic information about texts and the words they contain. Sometimes, morphological or syntactic information (such as the part-of-speech of words or the relations between words in a dependency grammar representation of sentences) is used to determine what keywords should be extracted. In some cases, certain PoS are given higher scores (e.g., nouns and noun phrases) since they usually contain more information about texts than other categories.

Some other methods make use of discourse markers (i.e., phrases that organize discourse into segments, such as however or moreover) or semantic information about the words (e.g. the shades of meaning of a given word). This paper can be a good introduction to how this information can be used in keyword extraction methods.

But, that's not all of the information you can use to extract keywords. Word co-occurrence can be used as well, e.g., the words that co-occur with topical words (as shown in this paper).

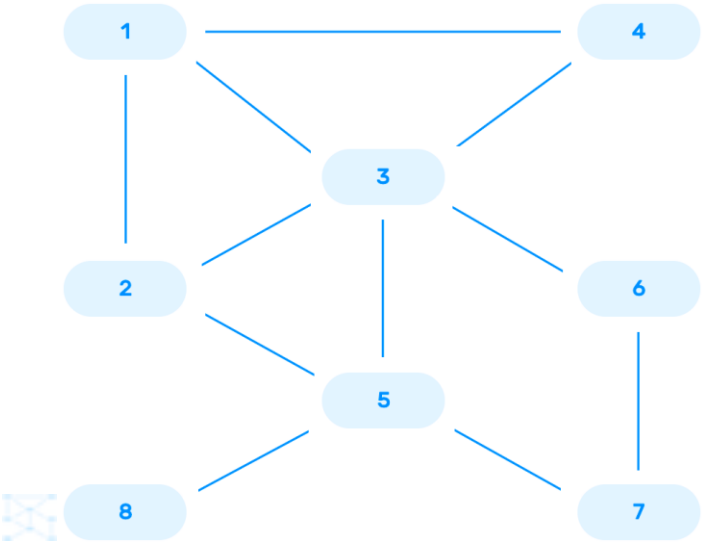
Most systems that use some kind of linguistic information outperform those that don't. We strongly recommend that you try some of them when extracting keywords from your texts.

Graph-based Approaches

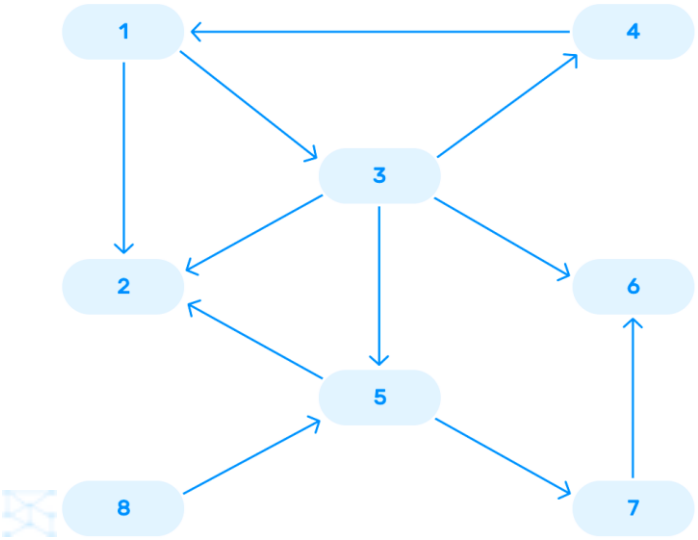
The most popular graph-based approach is the TextRank model, which we'll introduce later on in this post. A graph can be defined as a set of vertices with connections between them.

A text can be represented as a graph in different ways. Words can be considered vertices that are connected by a directed edge (i.e. a one-way connection between the vertices). Those edges can be labeled, for instance, as the relation that the words have in a dependency tree. Other representations of documents might make use of undirected edges, for example, when representing word co-occurrences.

If words were represented by numbers, an undirected graph would look like this:



A directed graph would look a little bit differently:

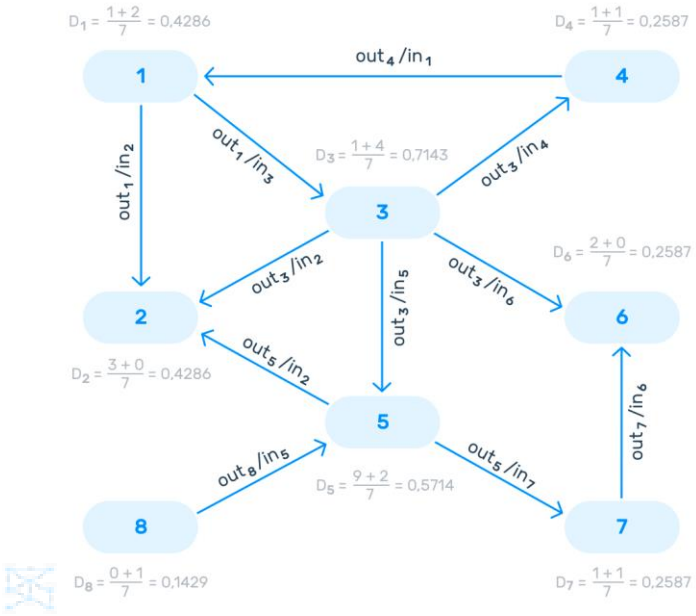


The underlying idea in graph-based keyword extraction is always the same: measuring how important a vertex is based on measures that take into consideration some information obtained from the structure of the graph to extract the most important vertices.

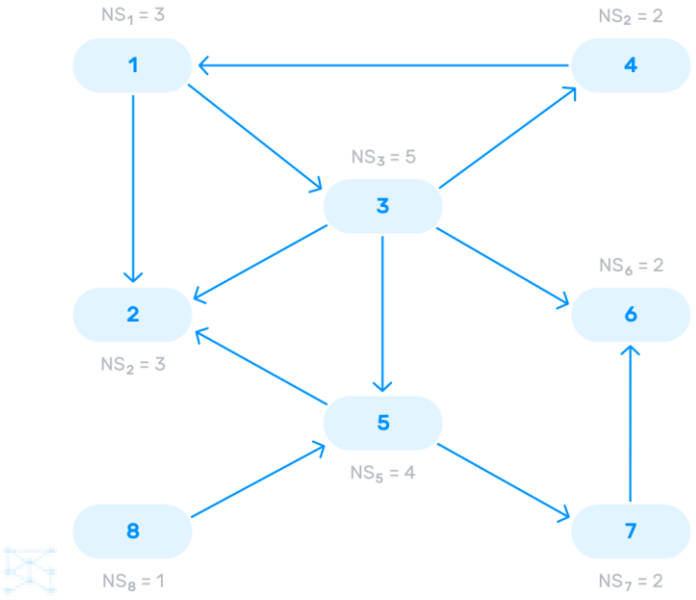
Once a graph has been built, it's time to determine how to measure the importance of the vertices. There are many different options, most of which are dealt with in this paper. Some methods choose to measure what is known as the degree of a vertex.

The degree of a vertex equals the number of edges or connections that land in the vertex (also known as the in degree) plus the number of edges that start in the vertex (also known as the out degree) divided by the maximum degree (which equals the number of vertices in the graph minus 1). This is the formula to calculate the degree of a vertex:

$$\mathbf{D_v = (D_v^{in} + D_v^{out}) / (N - 1)}$$



Some other methods measure the number of immediate vertices to a given vertex (which is known as neighborhood size).



No matter what the chosen measure is, there will be a score for each vertex that will determine whether it should be extracted as a keyword or not.

Take the following text as an example:

Automatic$_1$ graph-based$_2$ keyword$_3$ extraction$_4$ is pretty$_5$ straightforward$_6$. A document$_7$ is represented$_8$ as a graph$_9$ and a score$_{10}$ is given$_{11}$ to each of the vertices$_{12}$ in the graph$_{13}$. Depending$_{14}$ on the score$_{15}$ of a vertex$_{16}$, it might be chosen$_{17}$ as a keyword$_{18}$.

If we were to measure neighborhood size for the example above in a graph of dependencies which includes the content words only (numbered 1 - 18 in the text), the extracted keyphrase

would have been automatic graph-based keyword extraction since the neighborhood size of the head noun extraction (which equals 3 / 17) is the highest.

Machine Learning Approaches

Machine learning-based systems are used for many text analysis tasks, including keyword extraction. But what exactly is machine learning? It's a subfield of artificial intelligence that builds algorithms capable of learning from examples and making their own predictions.

In order to process unstructured text data, machine learning systems need to break it down into something they can understand. But how do machine learning models do this? By transforming data into vectors (a collection of numbers with encoded data), which contain the different features that are representative of a text.

There are different machine learning algorithms and techniques that can be used to extract the most relevant keywords in a text, including Support Vector Machines (SVM) and deep learning.

Below is one of the most common and effective approaches for keyword extraction with machine learning:

Conditional Random Fields

Conditional Random Fields (CRF) is a statistical approach that learns patterns by weighting different features in a sequence of words present in a text. This approach considers context and relationships between different variables in order to make its predictions.

Using conditional random fields allows you to create complex and rich patterns. Another advantage of this approach is its capacity to generalize: once the model has been trained with examples from a certain domain, it can easily apply what it has learned to other fields.

On the downside, in order to use conditional random fields, you need to have strong computational skills to calculate the weight of all the features for all the sequences of words.

Evaluating the Performance of Keyword Extractors

When it comes to evaluating the performance of keyword extractors, you can use some of the standard metrics in machine learning: accuracy, precision, recall, and F1 score. However, these metrics don't reflect partial matches; they only consider the perfect match between an extracted segment and the correct prediction for that tag.

Fortunately, there are some other metrics capable of capturing partial matches. An example of this is ROUGE.

ROUGE

ROUGE (recall-oriented understudy for gisting evaluation) is a family of metrics that compares different parameters (like the number of overlapping words) between the source text and the extracted words. The parameters include lengths and numbers of sequences and can be defined manually.

Hybrid Approaches

In order to get better results when extracting relevant keywords from text, you can combine two or more of the approaches that we've mentioned so far.

Now that we've learned about some of the different options available, it's time to see all the exciting things you can do with keyword extraction within a wide range of business areas, from customer support to social media management.

Use Cases and Applications of Keyword Extraction

Every day, internet users create 2.5 quintillion bytes of data. Social media comments, product reviews, emails, blog posts, search queries, chats, and so on. We have all sorts of unstructured text data at our disposal. The question is, how do we sort the chaos to find what's relevant?

Keyword extraction can help you obtain the most important keywords or key phrases from a given text without having to actually read a single line.

Whether you are a product manager trying to analyze a pile of product reviews, a customer service manager analyzing customer interactions, or a researcher that has to go through hundreds of online papers about a specific topic, you can put keyword extraction to use to easily understand what a text is about.

Thanks to keyword extraction, teams can be more efficient and take full advantage of the power of data. You can say goodbye to manual and repetitive tasks (saving countless human hours) and get access to interesting insights that will help you transform unstructured data into valuable knowledge.

Wondering what you can analyze keyword extraction? Here are some common use cases and applications:

1. Social media monitoring
2. Brand monitoring
3. Customer service
4. Customer feedback
5. Business intelligence
6. Search engine optimization (SEO)
7. Product analytics
8. Knowledge management

Background

In research & news articles, keywords form an important component since they provide a concise representation of the article's content. Keywords also play a crucial role in locating the article from information retrieval systems, bibliographic databases and for search engine optimization. Keywords also help to categorize the article into the relevant subject or discipline.

Conventional approaches of extracting keywords involve manual assignment of keywords based on the article content and the authors' judgment. This involves a lot of time & effort and also may not be accurate in terms of selecting the appropriate keywords. With the emergence of Natural Language Processing (NLP), keyword extraction has evolved into being effective as well as efficient.

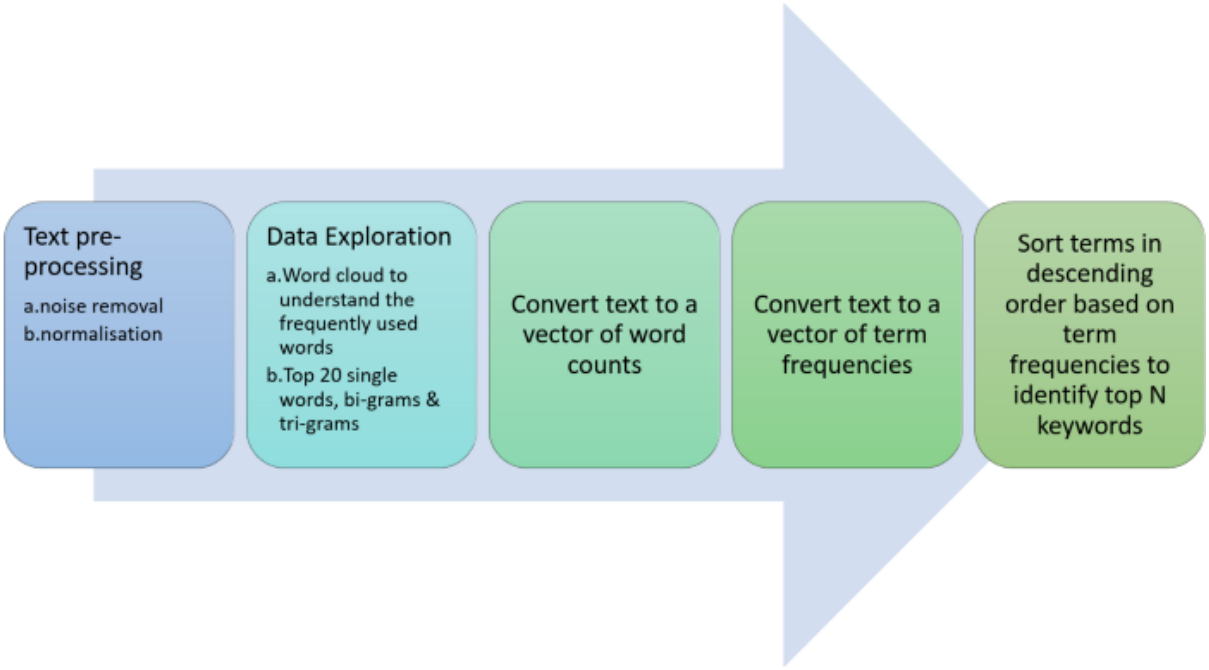
Dataset

In this article, we will be extracting keywords from a dataset that contains about 3,800 abstracts. The original dataset is from Kaggle — NIPS Paper. Neural Information Processing Systems (NIPS) is one of the top machine learning conferences in the world. This dataset includes the title and abstracts for all NIPS papers to date (ranging from the first 1987 conference to the current 2016 conference).

The original dataset also contains the article text. However, since the focus is on understanding the concept of keyword extraction and using the full article text could be computationally intensive, only abstracts have been used for NLP modelling. The same code block can be used on the full article text to get a better and enhanced keyword extraction.

Working

High-level approach



Importing the dataset

The dataset used for this article is a subset of the papers.csv dataset provided in the NIPS paper datasets on Kaggle. Only those rows that contain an abstract have been used. The title and abstract have been concatenated after which the file is saved as a tab separated *.txt file.

	id	year	abstract1
0	1861	2000	Algorithms for Non-negative Matrix Factorizati...
1	1975	2001	Characterizing Neural Gain Control using Spike...
2	3163	2007	Competition Adds Complexity It is known that d...
3	3164	2007	Efficient Principled Learning of Thin Junction...
4	3167	2007	Regularized Boost for Semi-Supervised Learning...

As we can see, the dataset contains the article ID, year of publication and the abstract.

Preliminary text exploration

Before we proceed with any text pre-processing, it is advisable to quickly explore the dataset in terms of word counts, most common and most uncommon words.

Fetch word count for each abstract

	abstract1	word_count
0	Algorithms for Non-negative Matrix Factorizati...	112
1	Characterizing Neural Gain Control using Spike...	88
2	Competition Adds Complexity It is known that d...	70
3	Efficient Principled Learning of Thin Junction...	150
4	Regularized Boost for Semi-Supervised Learning...	124

```
count    3847.000000
mean      155.710684
std        46.080919
min        27.000000
25%       122.000000
50%       150.000000
75%       185.000000
max       325.000000
Name: word_count, dtype: float64
```

The average word count is about 156 words per abstract. The word count ranges from a minimum of 27 to a maximum of 325. The word count is important to give us an indication of the size of the dataset that we are handling as well as the variation in word counts across the rows.

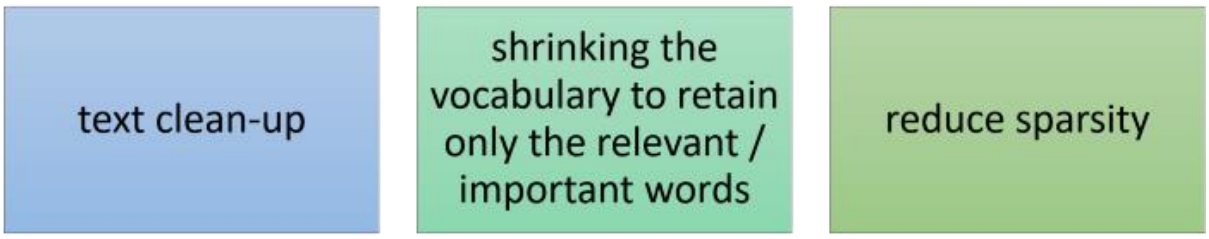
Most common and uncommon words

A peek into the most common words gives insights not only on the frequently used words but also words that could also be potential data specific stop words. A comparison of the most common words and the default English stop words will give us a list of words that need to be added to a custom stop word list.

```
the          29516
of           21238
a            16181
and          13882
to           12832
in            9272
for           8219
that          7680
is            7518
We            6112
on            5577
we            5064
with          5015
as            3608
this          3603
are           3458
an            3323
by            3237
can           2891
learning      2844
dtype: int64
```

Most common words

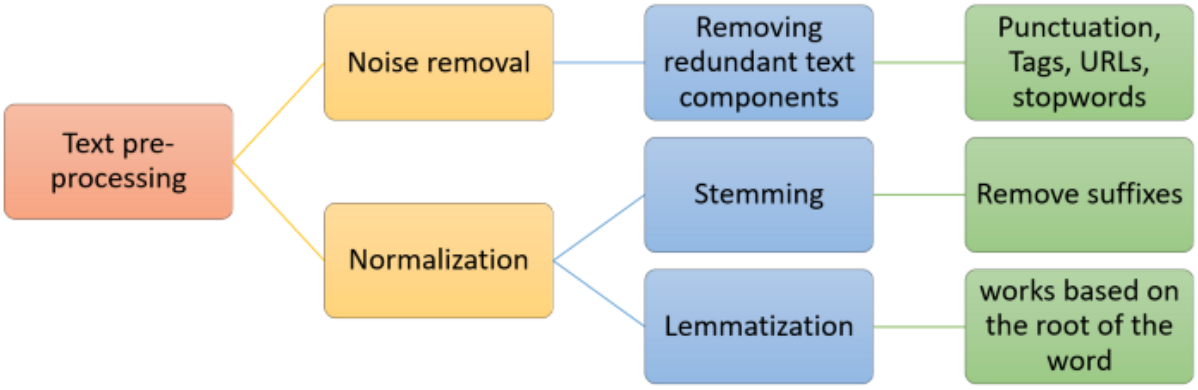
Text pre-processing



Objectives of text pre-processing

Sparsity: In text mining, huge matrices are created based on word frequencies with many cells having zero values. This problem is called sparsity and is minimized using various techniques.

Text pre-processing can be divided into two broad categories — noise removal & normalization. Data components that are redundant to the core text analytics can be considered as noise.



Text pre-processing

Handling multiple occurrences / representations of the same word is called normalization. There are two types of normalization — stemming and lemmatization. Let us consider an example of various versions of the word learn — learn, learned, learning, learner. Normalisation will convert all these words to a single normalised version — "learn".

Stemming normalizes text by removing suffixes.

Lemmatisation is a more advanced technique which works based on the root of the word.

The following example illustrates the way stemming and lemmatisation work:

```
stemming: invers
lemmatization: inversely
```

To carry out text pre-processing on our dataset, we will first import the required libraries.

Removing stopwords: Stop words include the large number of prepositions, pronouns, conjunctions etc in sentences. These words need to be removed before we analyse the text, so that the frequently used words are mainly the words relevant to the context and not common words used in the text.

There is a default list of stopwords in python nltk library. In addition, we might want to add context specific stopwords for which the "most common words" that we listed in the beginning will be helpful. We will now see how to create a list of stopwords and how to add custom stopwords:

We will now carry out the pre-processing tasks step-by-step to get a cleaned and normalised text corpus:

Let us now view an item from the corpus:

```
'extended level method efficient multiple kernel learning consider problem multiple kernel learning mkl formulated convex conca
ve problem past efficient method e semi infinite linear programming silp subgradient descent sd proposed scale multiple kernel
learning despite success method shortcoming sd method utilizes gradient current solution b silp method regularize approximate s
olution obtained cutting plane model work extend level method originally designed optimizing non smooth objective function conv
ex concave optimization apply multiple kernel learning extended level method overcomes drawback silp sd exploiting gradient com
puted past iteration regularizing solution via projection level set empirical study eight uci datasets show extended level meth
od significantly improve efficiency saving average computational time silp method sd method'
```

Data Exploration

We will now visualize the text corpus that we created after pre-processing to get insights on the most frequently used words.

Word cloud

Text Preparation

Text in the corpus needs to be converted to a format that can be interpreted by the machine learning algorithms. There are 2 parts of this conversion — Tokenisation and Vectorisation.

Tokenisation is the process of converting the continuous text into a list of words. The list of words is then converted to a matrix of integers by the process of vectorisation. Vectorisation is also called feature extraction.

For text preparation we use the **bag of words model** which ignores the sequence of the words and only considers word frequencies.

Creating a vector of word counts

As the first step of conversion, we will use the CountVectoriser to tokenise the text and build a vocabulary of known words. We first create a variable “cv” of the CountVectoriser class, and then evoke the fit_transform function to learn and build the vocabulary.

Let us now understand the parameters passed into the function:

- cv=CountVectorizer(max_df=0.8,stop_words=stop_words, max_features=10000, ngram_range=(1,3))

- max_df — When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). This is to ensure that we only have words relevant to the context and not commonly used words.

- max_features — determines the number of columns in the matrix.

- n-gram range — we would want to look at a list of single words, two words (bi-grams) and three words (tri-gram) combinations.

An encoded vector is returned with a length of the entire vocabulary.

```
['algorithm',
 'non',
 'negative',
 'matrix',
 'factorization',
 'nmf',
 'useful',
 'decomposition',
 'multivariate',
 'data']
```

Visualize top N uni-grams, bi-grams & tri-grams

We can use the CountVectoriser to visualise the top 20 unigrams, bi-grams and tri-grams.

Conversion to matrix

The next step of refining the word counts is using the TF-IDF vectoriser. The deficiency of a mere word count obtained from the countVectoriser is that, large counts of certain common words may dilute the impact of more context specific words in the corpus. This is overcome by the TF-IDF vectoriser which penalizes words that appear several times across the document.

TF-IDF are word frequency scores that highlight words that are more important to the context rather than those that appear frequently across documents.

TF-IDF consists of 2 components:

- TF — term frequency
- IDF — Inverse document frequency

- $TF = \frac{\text{Frequency of a term in a document}}{\text{total number of terms in the document}}$

- $IDF = \frac{\log(\text{Total documents})}{\text{\# of documents with the term}}$

Based on the TF-IDF scores, we can extract the words with the highest scores to get the keywords for a document.

Code

https://colab.research.google.com/drive/17Zv5AJXc_s2D2EiaQxYr_Oko3gr_6M5r?usp=sharing

Output

```
Abstract:
hierarchical learning dimensional bias human categorization existing model categorization typically represent classified item p
oint multidimensional space mathematical point view infinite number basis set used represent point space choice basis set psych
ologically crucial people generally choose basis dimension strong preference generalize along ax dimension diagonally make choi
ce dimension special explore idea dimension used people echo natural variation environment specifically present rational model
assume dimension learns type dimensional generalization people display bias shaped exposing model many category structure hypot
hesized like child encounter model viewed type transformed dirichlet process mixture model learning base distribution dirichlet
process allows dimensional generalization learning behaviour model capture developmental shift roughly isotropic child axis ali
gned generalization adult

Keywords:
dimension 0.301
people 0.268
basis 0.234
child 0.205
categorization 0.193
```

Conclusion

To be effective, the IDF computation should be based on a big corpus that is representative of the text for which the keywords need to be retrieved. The IDF extraction would be considerably more efficient in our case if we used the whole paper content instead of the abstracts. However, due to the large size of the dataset, I have reduced the corpus to only the abstracts for demonstration purposes.

This is a straightforward method to grasping basic NLP ideas and getting some hands-on practise with some Python code on a real-world use case. Keywords may be extracted from news feeds and social media feeds using the same method.