Contents

[Topic 1: Natural Language Processing I 5](#_Toc97059462)

[Natural language processing 5](#_Toc97059463)

[1.3 Natural language processing 5](#_Toc97059464)

[Components of NLP 5](#_Toc97059465)

[Natural language processing stages 6](#_Toc97059466)

[Lexical analysis 6](#_Toc97059467)

[Syntactic analysis (Parsing) 7](#_Toc97059468)

[Semantic analysis 7](#_Toc97059469)

[Pragmatic analysis 8](#_Toc97059470)

[Discourse analysis 8](#_Toc97059471)

[1.4 Why use natural language processing? 9](#_Toc97059472)

[Introduction to text mining 9](#_Toc97059473)

[1.5 Text mining and analytics 9](#_Toc97059474)

[Types of text mining 10](#_Toc97059475)

[Functions of text analytics 10](#_Toc97059476)

[1.6 Text pre-processing 12](#_Toc97059477)

[Basic pre-processing 12](#_Toc97059478)

[1.7 Basic pre-processing I 12](#_Toc97059479)

[Lowercasing 13](#_Toc97059480)

[1.8 Basic pre-processing II 13](#_Toc97059481)

[Noise Removal 13](#_Toc97059482)

[Common word removal 14](#_Toc97059483)

[1.9 Basic pre-processing III 14](#_Toc97059484)

[Quiz 15](#_Toc97059485)

[Topic 2: Natural Language Processing II 17](#_Toc97059486)

[Advanced text mining 17](#_Toc97059487)

[2.2 Introduction to text mining feature engineering 17](#_Toc97059488)

[2.3 N-grams 17](#_Toc97059489)

[2.4 Term frequency 18](#_Toc97059490)

[Inverse document frequency 19](#_Toc97059491)

[2.5 Inverse document frequency 19](#_Toc97059492)

[2.6 Term frequency–inverse document frequency 20](#_Toc97059493)

[2.7 Bag-of-words 22](#_Toc97059494)

[Word embeddings 22](#_Toc97059495)

[2.8 Word embeddings I 22](#_Toc97059496)

[Frequency-based embedding 23](#_Toc97059497)

[Prediction-based embedding 23](#_Toc97059498)

[Count Vector 23](#_Toc97059499)

[2.9 Word embedding II 24](#_Toc97059500)

[CBOW vs Skip-Gram 25](#_Toc97059501)

[2.10 Sentiment analysis and opinion mining I 26](#_Toc97059502)

[Subjective Analysis 27](#_Toc97059503)

[Individual perspective 27](#_Toc97059504)

[Business Perspective 27](#_Toc97059505)

[Applications 28](#_Toc97059506)

[2.11 Sentiment analysis and opinion mining II 28](#_Toc97059507)

[Opinion Mining 28](#_Toc97059508)

[Opinion mining sub-fields 29](#_Toc97059509)

[Sentiment classification 29](#_Toc97059510)

[Sentence level classification 29](#_Toc97059511)

[Document level classification 30](#_Toc97059512)

[Feature level classification 30](#_Toc97059513)

[Quiz 31](#_Toc97059514)

[What is an image? 33](#_Toc97059515)

[3.2 What is an image? 33](#_Toc97059516)

[What is an Image? 33](#_Toc97059517)

[Monochrome (1-bit) Image 34](#_Toc97059518)

[2-bit image 34](#_Toc97059519)

[Analogue Images 34](#_Toc97059520)

[Digital Images 34](#_Toc97059521)

[Image processing 36](#_Toc97059522)

[3.3 What is digital image processing? 36](#_Toc97059523)

[Why do we do it? 37](#_Toc97059524)

[Image pre-processing vs image processing enhancement 37](#_Toc97059525)

[3.4 Digital image processing cycle 38](#_Toc97059526)

[Image acquisition 38](#_Toc97059527)

[Image pre-processing 39](#_Toc97059528)

[3.5 Digital image processing tools 39](#_Toc97059529)

[3.6 Colour transformation 40](#_Toc97059530)

[3.7 Standardisation 41](#_Toc97059531)

[3.8 Image thresholding 41](#_Toc97059532)

[3.9 Image data augmentation 42](#_Toc97059533)

[3.10 Image smoothing 44](#_Toc97059534)

[3.11 Morphological transformations 45](#_Toc97059535)

[3.12 Sharpening an Image 45](#_Toc97059536)

[Canny Edge Detection 45](#_Toc97059537)

[Quiz 46](#_Toc97059538)

[Topic 4: Introduction to Graph Data 48](#_Toc97059539)

[Introduction to graph data 48](#_Toc97059540)

[4.2 Intro to graph data 48](#_Toc97059541)

[What is a graph? 48](#_Toc97059542)

[Where would we find graphs? 48](#_Toc97059543)

[When would we use graphs? 48](#_Toc97059544)

[Why do we use graphs? 49](#_Toc97059545)

[What is a graph? 49](#_Toc97059546)

[4.3 What is a graph? 49](#_Toc97059547)

[Directed Vs Undirected Graphs 50](#_Toc97059548)

[4.4 Where are graphs used? 50](#_Toc97059549)

[4.5 Graph vs structured data 51](#_Toc97059550)

[Converting graph data to a matrix 53](#_Toc97059551)

[4.6 Converting graph data into a matrix 53](#_Toc97059552)

[4.7 Applying Eigenvalues and Eigenvectors to graphs 54](#_Toc97059553)

[What are Eigenvalues and Eigenvectors? 54](#_Toc97059554)

[What do they tell us? 55](#_Toc97059555)

[Worked example of calculating Eigenvalues and Eigenvectors for a given matrix 55](#_Toc97059556)

[K-Means 57](#_Toc97059557)

[4.8 K-Means recap 57](#_Toc97059558)

[Spectral clustering 57](#_Toc97059559)

[4.9 Spectral clustering 57](#_Toc97059560)

[What is Spectral clustering? 57](#_Toc97059561)

[Steps to spectral clustering 58](#_Toc97059562)

[4.10 K-medoids clustering 59](#_Toc97059563)

[Partitioning Around Medoids (PAM) 59](#_Toc97059564)

[4.11 Spectral clustering worked example 60](#_Toc97059565)

[Quiz 60](#_Toc97059566)

Topic 1: Natural Language Processing I

# Topic 1: Natural Language Processing I

# Natural language processing

## 1.3 Natural language processing

This step will provide a brief introduction to Natural Language Processing (NLP).

NLP is primarily concerned with getting computers to perform useful and interesting tasks with human languages. Secondly, it is concerned with helping us come to a better understanding of human language. For me, the topic is quite a change from normal data mining processes, so you may find it useful to refer to Speech And Language Processing by Daniel Jurafsky and James H. Martin (2000). In the next three steps, we are going to cover quite a bit of terminology. This is important as it is constantly referred to in the literature.

In order to process written text, we need:

* Lexical, syntactic, and semantic knowledge
* Discourse information
* Real-world knowledge

Additionally, for speech content, we would require more knowledge such as:

* Speech synthesis
* Challenges of speech recognition

### Components of NLP

There are two major components of NLP:

* Natural language understanding
* Natural language generation

**Natural Language Understanding (NLU)** is a component that is becoming popular these days through products such as Alexa, Siri and Google Assistant. It is a type of NLP that covers the reading aspect of NLP. The common applications are:

* Simple profanity filters
* Sentiment detection
* Topic classification
* Entity detection

It involves tasks such as:

* Mapping the natural language input to useful representation.
* Analysing different aspects of the language.

**Natural Language Generation (NLG)** is the process of generating content from knowledge bases. It involves constructing phrases and sentences in the form of natural language.

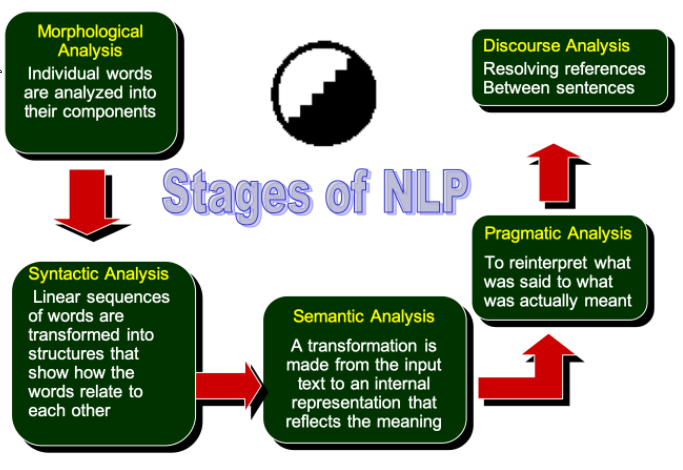
The tasks involved are:

* **Text planning:** Retrieve relevant content from the knowledge base.
* **Sentence planning:** It includes processes such as choosing the right words, form phrases and setting the tone of the sentence.
* **Text realization:** Structure the sentence plan.

### Natural language processing stages

NLP consists of five stages. They are:

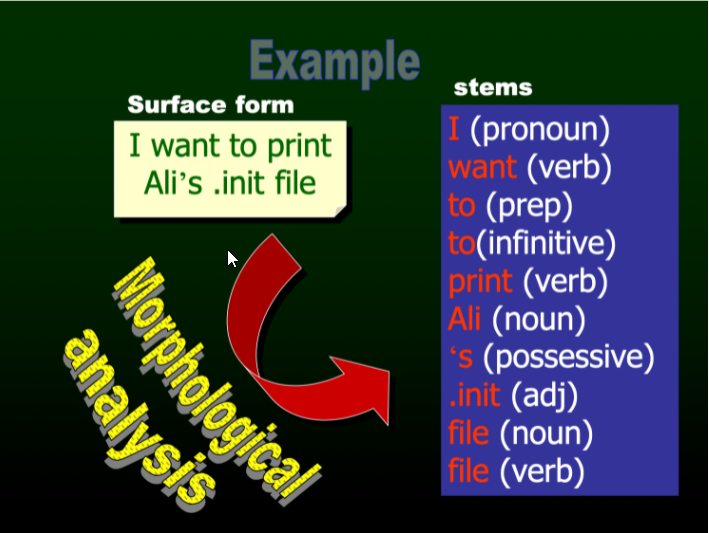
1. Lexical analysis (morphological analysis)
2. Syntactic analysis
3. Semantic analysis
4. Pragmatic analysis
5. Discourse analysis



Let’s look at the stages in detail below:

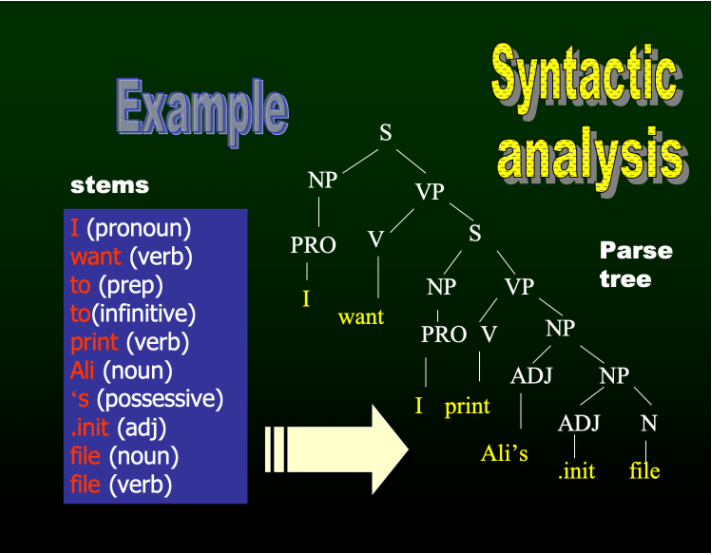
### Lexical analysis

In Lexical Analysis, individual words are identified and categorised. It breaks down the text into chunks such as paragraphs, words, etc.



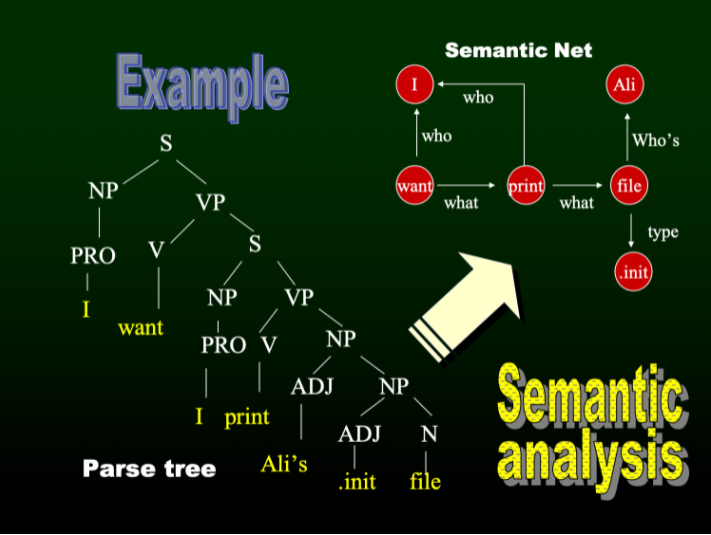
### Syntactic analysis (Parsing)

Syntactic analysis, also called Parser Analysis, checks the extracted words for grammar. It studies the structure of the sentence or paragraph and learns how it relates to each other, creating a parse tree.



### Semantic analysis

Semantic analysis maps the words to the exact dictionary meaning. It checks for meaningfulness and disregards meaningless sentences or phrases, such as “hot ice-cream”.

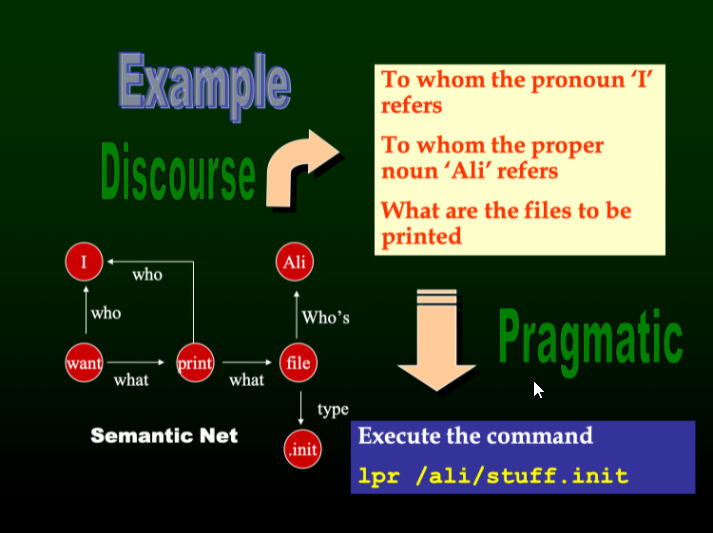


### Pragmatic analysis

Pragmatic analysis is the study of use of language in social context. It gets the meaning from Semantic Analysis, and re-interprets what is actually means. An example of Pragmatics is how same words can have different meaning in different settings. An example of this is the word bat, which can mean the animal or the piece of sporting equipment.

### Discourse analysis

Discourse analysis interprets the meaning of a sentence by understanding the preceding sentence. Consequently, it also interprets the succeeding sentence. It resolves the difference between sentences and provides the output combining with pragmatic analysis.



## 1.4 Why use natural language processing?

This step will highlight where Natural Language Processing (NLP) can be used.

NLP is widely used in our lives today in various applications such as:

* **Virtual assistants:** Virtual assistants, such as Siri and Alexa, can process human speech and perform actions, such as play music and order things online.
* **Chatbots:** Chatbots are used by industries to communicate with their customers catering to their needs. For example, online customer service nowadays is handled by chatbots at the beginning of a conversation. This conversation is then later handed over to service personnel if required.
* **Translation:** Translation services are available to convert written text and speech from one language to another. Google Translate is one such example, which can identify content through voice, text, and handwriting.
* **Social media monitoring:** Twitter and Facebook use NLP to monitor posts for profanity and take necessary action if any is found.
* **Advertisement matching:** Industries use NLP to recommend advertisements to users, based on their history.
* **Information extraction:** Information extraction is used to find content from a database of texts, to extract information from messages or articles and to summarise text.
* **Weather report generation:** A computer program in Canada accepts daily weather data and automatically generates reports in English and French.

Other examples include:

* **Spell Checking** of Documents
* **Information Retrieval**

There are many more applications under research and development such as:

Grading student essays.

Automated tutor intervention when a user makes a mistake through speech.

Monitor soccer games and preparing reports.

Predicting upcoming words and expanding abbreviations to help people with disabilities to communicate.

# Introduction to text mining

## 1.5 Text mining and analytics

Advanced text mining uses sophisticated NLP algorithms. It converts unstructured text data into meaningful data for analysis. It studies the text from various sources such as feedback and customer opinions. Text mining provides discernible data to improve decisions and predictions.

It is gaining momentum in various applications such as:

* Sentiment analysis on social media
* Predicting stock market
* Predicting churn
* Customer Influence
* Customer service and helpdesk

### Types of text mining

The various types of text mining include:

* **Search and information retrieval:** Storage and retrieval of documents, including search engines and keyword search
* **Document clustering:** Grouping and categorizing terms, snippets, paragraphs, and documents using clustering methods
* **Document classification:** Grouping and categorizing snippets, paragraphs, and document using Data Mining Classification methods, based on methods trained on labelled examples
* **Web mining:** Data and text mining
* **Information extraction:** Identification and extraction of relevant facts and relationships from unstructured text
* **Natural language processing:** Low-level language processing and understanding of tasks
* **Concept extraction:** Grouping of words and phrases into semantically similar groups

### Functions of text analytics

There are seven basic functions of Text Analytics. They are:

1. Language identification
2. Tokenization
3. Sentence breaking
4. Part of speech tagging
5. Chunking
6. Syntax parsing
7. Sentence chaining

Let’s look at each function in detail below:

#### Language identification

There are over six thousand languages with their own unique characteristics. Hence, the first step is for the system to identify the language of the input text. The language identification process is most essential because it determines the process for all other text analysis functions.

#### Tokenization

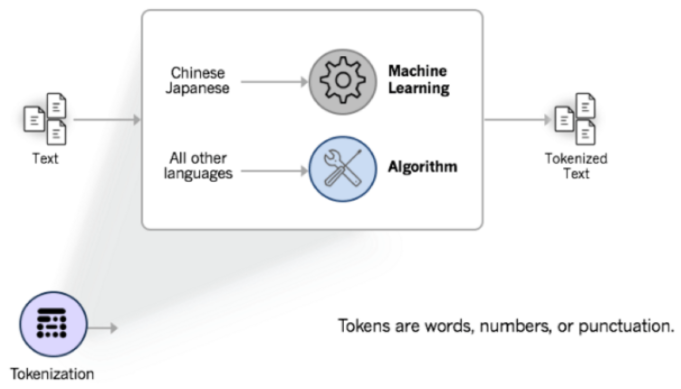
Tokenization is the process of breaking down the text into small parts. The splitting of text is done with the help of tokens which can be like:

* Punctuation marks
* Hyperlinks
* White space
* Possessive markers

Application of Tokenization varies with different languages:

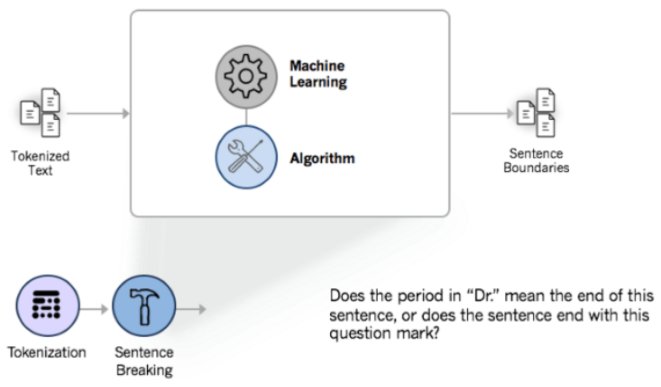
* **Alphabetic languages:** This follows straightforward conventions using the tokens described above. English is one such language where words are separated by white spaces or punctations which acts as tokens. Can user **algorithms**.
* **Logographic languages:** These types of languages could use drawings as characters which may not have white space between words, such as Chinese. These languages are processed with the help of **machine learning**.

Figure 1 shows a comparison of the tokenization processes for both the Alphabetic and Logographic languages.



#### Sentence breaking

Sentence breaking is the function of informing the system where the sentence ends after the text has been tokenized. It is a process of clearly defining boundaries of a sentence before deeper Text Analysis functions.



## 1.6 Text pre-processing

Text pre-processing is the fundamental and essential part of NLP as the characters, words and sentences identified at this stage makes up the initial input that is passed to all further processing stages.

Text pre-processing will start with analysing and tagging components such as **morphological analysis** and **PoS** (part of speech) tagging, through applications such as information retrieval and machine translation systems. It is a collection of activities in which text documents are pre-processed to **eliminate special formats** such as number format, date format, and **prepositions**, which are unlikely to help text mining. Before we get into some of the most common forms of text pre-processing techniques, we will look at a number of basic feature extraction techniques and they are as follows:

* Number of words
* Number of characters
* Average word length
* Number of stopwords
* Number of special characters
* Number of numerics
* Number of uppercase words

We will now use Python code to examine and extract the above features from a dataset. We are going to use a sample of Twitter text and you can download it by clicking this link:

<https://www.computing.dcu.ie/~amccarren/mcm_data/train_E6oV3lV.csv>

As usual, mount a drive and put the data in the folder of your choosing.

Please go to this Google Colab link to complete this step:

<https://drive.google.com/open?id=1WOqwidohd7F7QBe2iU9-dX10o116MHUP>

# Basic pre-processing

## 1.7 Basic pre-processing I

As with any pre-processing techniques used in data analytics, we need to convert our text data into a form that will allow us to make it as easy as possible to extract insights from it.

There are many different ways to pre-process text. Text pre-processing is highly application dependent. For example, we may be using a Twitter database and remove strange characters by converting “new” words to words that are in a common dictionary (e.g. this is known as noise removal and normalization). Text normalisation is the process of transforming text to a canonical form. For example, we may convert “2morrow” and “tomorw” to “tomorrow”. This paper shows us how normalisation can be useful in determining Twitter sentiment:

<https://sentic.net/microtext-normalization.pdf>

Noise removal can be the removal of spurious characters or text that can interfere or take away from your analysis. We will cover a number of techniques that can help with both of these tasks in this step and the following steps:

**Step 1:**

* Lowercasing
* Removing Punctuation
* Stopword Removal

**Step 2:**

* Common word removal
* Rare words removal
* Spelling correction

**Step 3:**

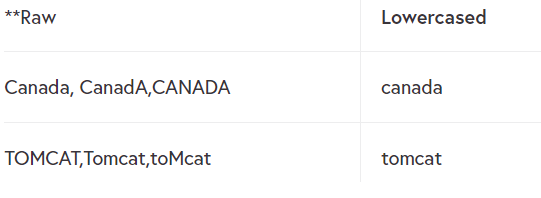
Following this we will look at reducing or word set down or categorizing words using:

* Stemming
* Lemmatization
* Tokenization

Let’s look at these approaches in detail in the following Steps using Python code.

### Lowercasing

Lowercasing is a technique to convert all of the text in a document to lowercase. It is one of the simplest and mo effective forms of Text Pre-Processing. It is applicable to most Text Mining and NLP problems. This process is done to avoid multiple copies of the same word in a document and also to reduce the vocabulary of the text data.



A Python example of the process along with the remainder of this step is available in the following Colab:

<https://drive.google.com/open?id=10YmwOkCym66YKpiiKbCw_VSbb028lnrN>

## 1.8 Basic pre-processing II

In the previous step, we discussed a number of simple pre-processing techniques. In this step, we will discuss a number of techniques that will reduce the dataset.

Sometimes you will find that particular words are recurring in a piece of text all the time. So if you want to know how people felt about the management of Manchester United then you would expect to see words like ManU, United etc in each tweet. In this case, you might remove these words as they are of no benefit in categorizing the text. Additionally, spelling mistakes can have a detrimental impact in text analysis, so standardizing the dataset by using some form of spelling corrector can be useful. We will now cover the following text processing activities.

### Noise Removal

* Common word removal
* Rare words removal
* Spelling correction

Let’s look at these techniques using python examples.

### Common word removal

In Step 1.6 we described stopwords and showed you how we can remove them. These types of words occur frequently. Words such as “a”, “and” etc… are typical stop words, and can typically be found in the NLTK stopwords library:

<https://www.nltk.org/>

<https://github.com/igorbrigadir/stopwords/blob/master/en/terrier.txt>

We can also remove commonly occurring words from our document collection. These may be specific to our subject, so they do not show up in the NLTK’s stopword dictionary. For example, we may examine our data for the ten most frequently occurring words. Then depending on the words and their context we can make a decision whether or not to remove them. These words could be considered to be stopwords for our document collection but would not necessarily be in the NLTK stopword list.

Follow this link to go to the Google Colab to complete this step:

<https://drive.google.com/open?id=1l2wFzzger_fCiwPuwbjGYcCqaZSIW4V->

## 1.9 Basic pre-processing III

We are going to use the description of the three ideas by Standford, which is as follows:

<https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html>

For grammatical reasons, documents are going to use different forms of a word, such as organize, organizes, and organizing. Additionally, there are families of derivationally related words with similar meanings, such as democracy, democratic, and democratization. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set.

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

* am, are, is be
* car, cars, car’s, cars’ car

A token is an instance of a sequence of characters in some particular document that is grouped together as a useful semantic unit for processing. Tokenization is the task of chopping it up into pieces, called tokens.

As usual, we are going to use Python to show what each technique does. Please follow this Colab link now.

<https://drive.google.com/open?id=1oXU2uaN361m5oI5BnlDVoZYg4HhqVm_6>

# Quiz

**Question 1 –** Natural language process covers text planning and sentence planning.

* True
* False

*These items are covered by Natural Language Generation (NLG).*

**Question 2 –** Pragmatic analysis interprets what was said to what was actually meant.

* True
* False

*Yes, pragmatic analysis interprets what was said to what was actually meant.*

**Question 3 –** Syntactic analysis is also known as sentiment analysis.

* True
* False

*No it is as Parser analysis.*

**Question 4 -** Tokenization is the process of combining text from different sources.

* True
* False

*Tokenization is the process of breaking down the text into small parts*

**Question 5 –** Sentence Breaking is the function of informing the system where the sentence ends after the text has been tokenized.

* True
* False

*Yes, sentence breaking is the function of informing the system where the sentence ends after the text has been tokenized.*

**Question 6 –** A stopword is a word that does not occur frequently.

* True
* False

*Stopwords are commonly occurring words*

**Question 7 –** Common words are another name for stop words.

* True
* False

*Stopwords are generally common words but not all common words are stopwords.*

**Question 8 –** Noise removal can be the removal of spurious characters or text that can interfere or take away from your analysis.

* True
* False

*Yes, noise removal can be the removal of spurious characters or text that can interfere or take away from your analysis.*

**Question 9 –** When we chop off the ends of words in the hope of achieving a standard word by removing derivational affixes is known as Lemmatization.

* True
* False

*The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.*

Topic 2: Natural Language Processing II

# Topic 2: Natural Language Processing II

# Advanced text mining

## 2.2 Introduction to text mining feature engineering

This step will detail the methods used in text mining applications.

In the previous topic, we covered a number of text “cleaning” and “enhancement” techniques. These are very useful when initially pre-processing text data and do help in reducing the size of the problem. However, if we want to build complex text-based prediction algorithms, then we are going to have to create a number of more advanced engineered features from our data. These techniques are used in many text mining applications. Specifically, the methods we will use in this topic are:

* Ngrams
* Term frequency
* Inverse document frequency
* Term frequency – Inverse document frequency (TF-IDF)
* Bag of words
* Word Embeddings

Many of you will also have heard of a concept known as sentiment analysis, this topic is an engineered feature but is also an incredibly useful technique when trying to understand the tone of a piece of text. We will also spend some time on this topic.

In the next step, we will discuss Ngrams.

## 2.3 N-grams

In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of ***n*** items from a given sample of text or speech. The items can be:

phonemes, syllables, letters, words or base pairs according to the application…

The n-grams typically are collected from a text or speech corpus and are the combination of multiple words used together. When the items are words, n-grams may also be called *shingles*. Let us take a look at the following examples:

* *San Francisco* = 2-gram
* *The Three Musketeers* = 3-gram
* *She stood up slowly* = 4-gram

Now you will have seen each of these examples to varying extents. The first two more regularly than the third. **We can assign probabilities to the occurrence of a particular n-gram**. This will help us join words for further analysis purposes. The basic principle behind -grams is that they capture the language structure, such as what letter or word is likely to follow the given one. The longer the n-gram (the higher the n), the more context you have to work with. Optimum length really depends on the application:

If your n-grams are too short 🡺 you may fail to capture important differences

If your n-grams are too long 🡺 you may fail to capture the “general knowledge” and only stick to particular cases.

N-grams with N=1 are called *unigrams*.

N-grams with N=2 are called *bigrams*

N-grams with N=3 are called *trigrams*

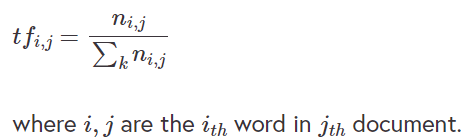
So, let’s quickly extract bigrams from our tweets using the -grams function of the textblob library. In the following example, we will show you how to create an n-gram and then use it to create a model which will give predictions for the next word. Go to this Goolge Colab now to complete this step:

<https://drive.google.com/open?id=1ALk6rGAw4slIw0_qXhtOLk-JUhZ_RPvK>

## 2.4 Term frequency

**TF** stands for “Term Frequency” and **counts how many times a word has occurred in a given corpus**.

Since a corpus is made up of many documents, each document and its words will have their own TF count. Term frequency indicates the significance of a particular term within the overall document. The calculation for document TF is given in the following equation.



TF gives us the frequency of the word in each document in the corpus. It is the ratio of the number of times the word appears in a document compared to the total number of words in that document. It increases as the number of occurrences of that word within the document increases and each document has its own TF.

Let’s take an example to get a clearer understanding.

* Sentence 1: *The car is driven on the road*.
* Sentence 2: *The truck is driven on the highway*.



***I think above is incorrect!? “The” should be TF 2/7???***

This article will give some additional help if you need it:

<https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/>

[VIDEO:] <https://www.youtube.com/watch?v=vZAXpvHhQow>

and is where we took the sample code. Please go to this Google Colab to complete this step.

<https://drive.google.com/open?id=1gA3J356U_I7bYiKv9RvE1Hf75rguoa4j>

# Inverse document frequency

## 2.5 Inverse document frequency

The intuition behind Inverse Document Frequency (IDF) is that a word is not of much use to us if it’s appearing in all the documents.

Therefore, the IDF of each word is the **log of the** **ratio of the total number of rows to the number of rows in which that word is present**.

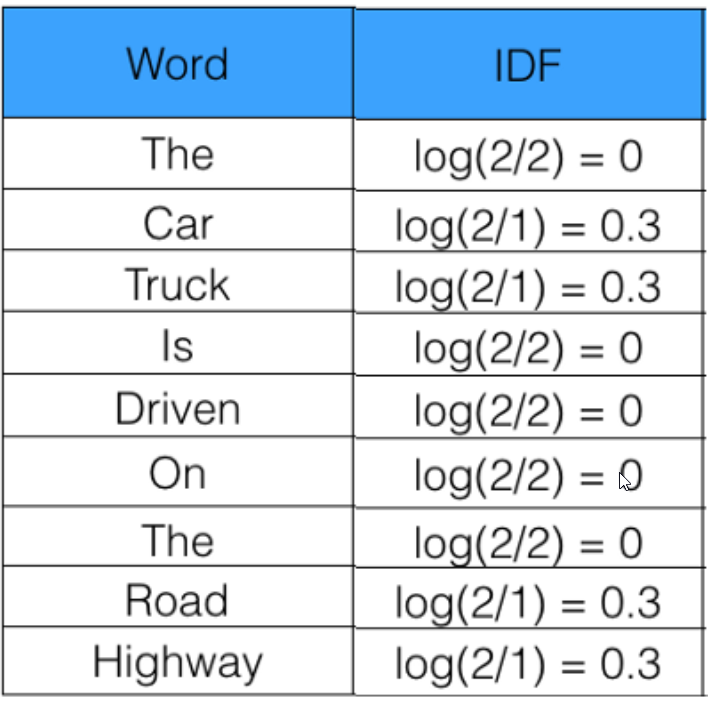


Where ***N*** is the total number of rows/documents and ***dfi*** is the number of documents containing word ***i***.

Let’s take an example to get a clearer understanding. So let’s use the sentences we had in the previous step:

* Sentence 1: The car is driven on the road.
* Sentence 2: The truck is driven on the highway.

So the IDF is calculated for each word.



So, let’s calculate IDF for the same sentences and tweets for which we calculated the term frequency. Follow this link to go to the Google Colab for this step:

<https://drive.google.com/open?id=1_C2-oI-JB6G3HUR4rL_K6HqVgu6hIrmT>

## 2.6 Term frequency–inverse document frequency

Term Frequency-Inverse Document Frequency or **TF-IDF** is a weight often used in information retrieval and text mining.

**This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus**.

* Importance increases proportionally to the number of times a word appears in the document.
* Importance is offset by the frequency of the word in the corpus.

The intuitive idea behind it is **a word is considered important to a document if it occurs a lot in the document but does not show up frequently in the corpus**. In this situation, we would want a high figure for the word in question. However, if the word occurs frequently in this document and all other documents then a low score would be required. So:

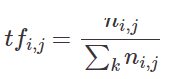
**TF** 🡺 Measures the importance of the word in a particular document.

**IDF** 🡺 Measures how frequently it is used in the corpus and penalizes words that occur frequently across all documents.

**TD-IDF** 🡺 Measures how important a word is to a document in a collection or corpus.

Variations of the TF-IDF weighting scheme are **often used by search engines** as a central tool in scoring and ranking a document’s relevance given a user query. TF-IDF can be successfully **used for stop-words filtering** in various subject fields including text summarization and classification.

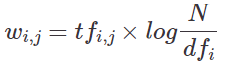
In Step 2.4 and Step 2.5 we outlined the formulas for TF and IDF as follows:



where ***i*** and ***j*** are the ***ith*** word in ***jth*** document.

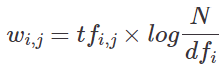


and



where, N is the total number of rows/documents and ***dfi*** is the number of documents containing word ***i***.

TF-IDF is the multiplication of the TF and IDF which we calculated above to give the following formula:



**Example:**

let’s consider:

* A document containing **100 words**
* Wherein the word **cat appears 3 times**.
* **TF** for cat is: **(3 / 100) = 0.03**.

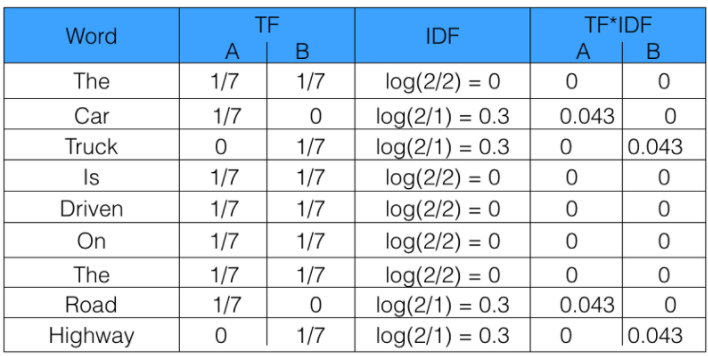
Now, assume we have:

* 10 million documents
* And the word cat appears in one 1,000 of these documents.
* **IDF** is calculated as: **log(10,000,000 / 1,000) = 4**.

Thus, the **TF-IDF** weight is the product of these quantities: **0.03 \* 4 = 0.12**.

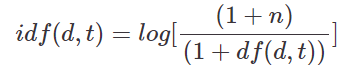
Figure 1 shows how we calculate the TF-IDF for the examples we looked at in the previous two steps:

* Sentence 1: *The car is driven on the road*.
* Sentence 2: *The truck is driven on the highway*.



***I think above is incorrect!? “The” should be TF 2/7???***

In the Google Colab below, let’s look at a Python example where we calculate the TF-IDF manually and then use skikit- learns “Tfidfvectorizer”. This is a very quick way to compute the TF-IDF for a corpus. The one thing to remember here is scikit-learn adds a “1” to the numerator and denominator of the IDF. This prevents zero divisions:



You can access the Google Colab for this step here:

<https://drive.google.com/open?id=1uMwy4RNSQfRwGqyCCjscXwHEe45UFED0>

## 2.7 Bag-of-words

A bag-of-words (BoW) model, is a way of extracting features from the text for use in modelling. The approach is very simple and flexible and can be used in a myriad of ways for extracting features from documents.

A BoW is a representation of text that describes the occurrence of words within a document. It involves two things:

1. A **vocabulary** of known words.
2. A **measure of the presence** of known words.

**BoW counts how many times a word appears in a document**. It’s a tally. Those word counts allow us to compare documents and gauge their similarities for applications like search, document classification and topic modelling. **BoW is also a method for preparing text for input in a deep-learning net**.

**BoW lists words paired with their word counts per document**. The words and documents in a text are converted into vectors, with **each element of the vector representing the occurrence of word in a document**. Each of the documents in the corpus is represented by columns of equal length.

The intuition is that documents are similar if they have similar content. Further, from the content alone we can learn something about the meaning of the document.

The BoW can be as simple or complex as you like. The complexity comes both in deciding how to design the vocabulary of known words (or tokens) and how to score the presence of known words.

Let’s have a look at an example in Python in this Google Colab here:

<https://drive.google.com/open?id=1jQlxlOiqH0h9ukBl7kx0Cm7VC8zHXtX0>

# Word embeddings

## 2.8 Word embeddings I

Word embedding is the collective name for a set of language modelling and feature learning techniques in natural language processing, where **words or phrases from the vocabulary are mapped to vectors of real numbers**. We can see examples of their use when we use sentiment analysis of reviews by Amazon etc., document or news classification or clustering by Google etc.

As it turns out, **many Machine Learning algorithms and almost all Deep Learning Architectures are incapable of processing strings or plain text in their raw form**. They require numbers as inputs to perform any sort of job. And with the huge amount of data that is present in text format, it has become ever more important to extract knowledge out of it and build applications. Some real-world applications of text applications are sentiment analysis of reviews by Amazon, news classification or clustering by Google. You can read more about this here:

<https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/>

Word embedding generally tries to map a word using a dictionary to a vector. For example, if we look at this example sentence:

* *‘Word Embeddings are Word converted into numbers’*

If we create a dictionary of all the unique words in the sentence, it looks like this:

* ‘Word’,’Embeddings’,’are’,’Converted’,’into’,’numbers’.

A vector representation of a word may be a one-hot encoded vector where 1 stands for the position where the word exists and 0 everywhere else. Taking dictionary above, the vector representation of:

* “numbers” 🡺 [0,0,0,0,0,1]
* “converted” 🡺 [0,0,0,1,0,0]

This is just a very simple method to represent a word in the vector form.

There are generally two types of word embeddings in NLP and they can be categorized as follows:

1. Frequency-based embedding
2. Prediction-based embedding

### Frequency-based embedding

There are generally three kinds of frequency-based approaches:

1. Count Vector
2. TF-IDF Vector
3. Co-Occurrence Vector

We have already used TF-IDF and Term frequency to create word vectors that we could use for prediction. We haven’t come across Count Vectors or Co-Occurrence Vectors before so we will briefly review them.

### Prediction-based embedding

When examining prediction-based word embedding we will look at the following:

* CBOW (Continuous Bag of words)
* Skip – Gram model

In the file link below, we will go to a Python notebook and have a look at frequency-based embedding followed by predictive embedding.

### Count Vector

The first embedding technique we are going to look at is using frequency-based Count Vector.

This creates a vector with each word as a position in the vector. The occurrence of the word in the vector. Let us do a quick example with the following two sentences:

* *He is a lazy boy. She is also lazy.*
* *Neeraj is a lazy person.*

The dictionary for this example are unique tokens (i.e. words) in the corpus:

[‘He’,’She’,’lazy’,’boy’,’Neeraj’,’person’]

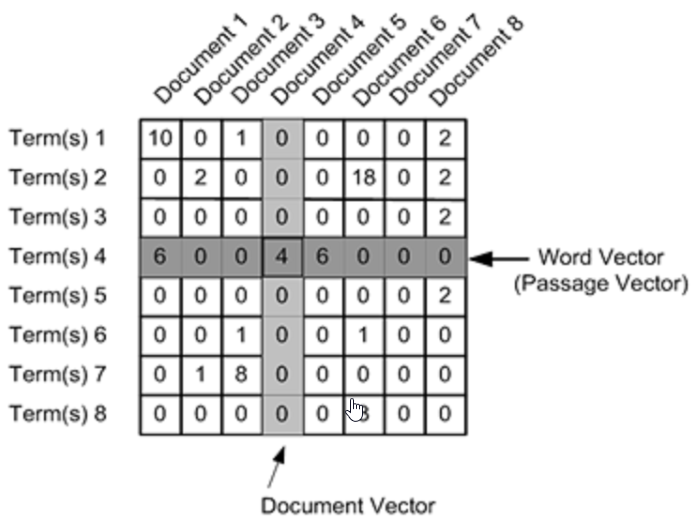
and the corresponding Matrix M can be seen to represent the vectors for each word.



Each column is a word vector for the corresponding word in the matrix M. For example, the word vector for ‘lazy’ in the above matrix is [2,1] and so on. Here, the rows correspond to the documents in the corpus and the columns correspond to the tokens in the dictionary. The second row in the above matrix may be read as D2 contains ‘lazy’: once, ‘Neeraj’: once and ‘person’ once.

In the real-world application, we might have a corpus which contains millions of documents. With millions of documents, we can extract hundreds of millions of unique words. So basically, the matrix will be very sparse. An alternative to using every unique word as a dictionary element would be to pick say top 10,000 words based on the frequency and then prepare a dictionary. To calculate the numbers the elements of the word vector we could either take the frequency (number of times a word has appeared in the document) or the presence (has the word appeared in the document?) to be the entry in the word vector. Generally, the frequency method is preferred over the latter.

Figure 1 shows a representational image of the matrix M for easy understanding.



Now, follow this link to go to the Google Colab file to complete this step:

<https://drive.google.com/open?id=12OaBXpqxOcvjUph120njQn2sfQOdpT6u>

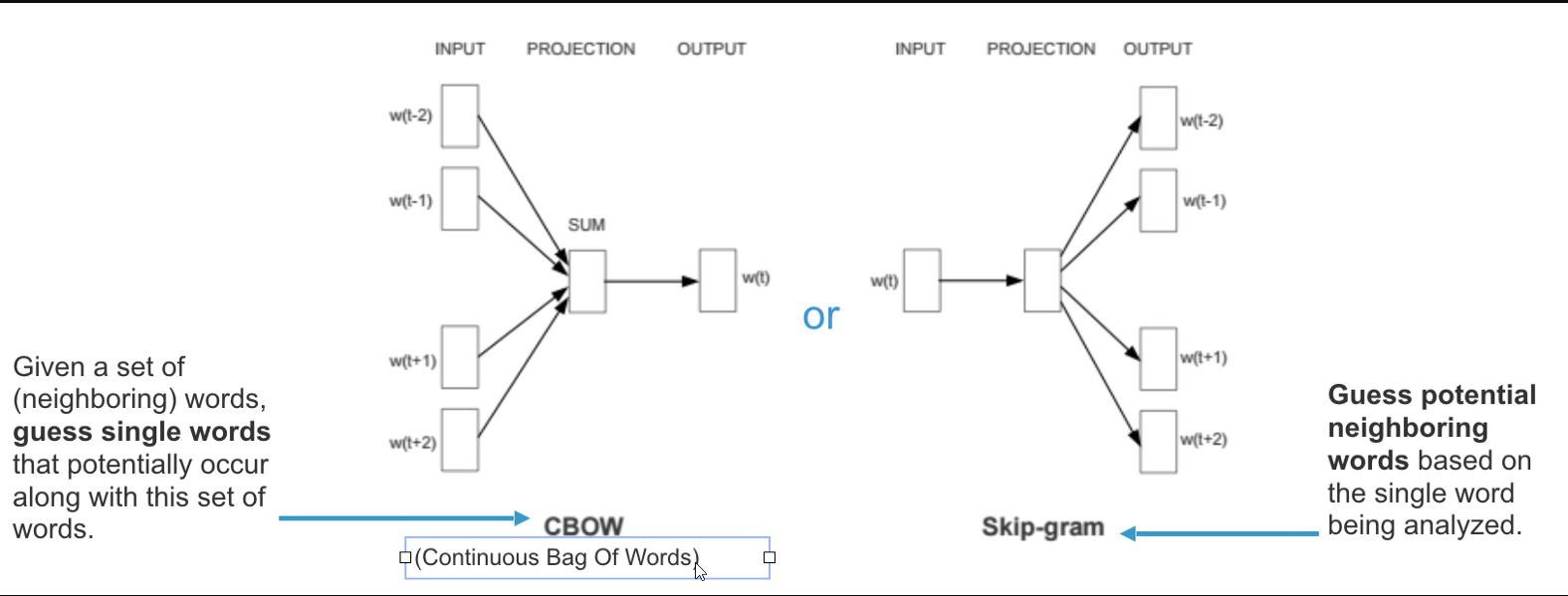
## 2.9 Word embedding II

In the previous step, we used frequency counts which were the aggregation of one hot encoded descriptions of each word in our document.

Although one-hot representation is simple, there are weak points in that it is impossible to obtain meaningful results with arithmetic between vectors. Let’s say we take an inner product to calculate the similarity between words. In one-hot representation, different words are 1 in different places and the other elements are 0. Thus, the result of taking the dot product between the different words is 0. This is not a useful result. We will want to do this in some form to reflect the level of similarity between words or sentences. The other problem we have with one-hot encoding is the number of dimensions. Generally, in most practical cases if we use one-hot encoding we will generate an enormous amount of sparse matrices.

In order to address this problem, it would be useful if we could create vectors that would reflect the similarity between words and also reduce the dimensionality of our dataset. So for example, if we wanted to represent a number of words by a single word, then predictive word embedding is appropriate.

In 2013 Word2vec was created, published and patented by a team of researchers led by Tomas Mikolov at Google. Word2Vec consists of models for generating word embedding. These models are shallow two-layer neural networks having one input layer, one hidden layer and one output layer. Word2Vec utilizes two architectures as can been seen in Figure 1 below:



### CBOW vs Skip-Gram

**CBOW** (Continuous Bag of Words): CBOW model **predicts the current word given context words within a specific window**.

* **Input layer** contains the context words.
* **Output layer** contains the current word.
* **Hidden layer** contains the number of dimensions in which we want to represent the current word present at the output layer.

CBOW is several times faster to train than the skip-gram and has slightly better accuracy for the frequent words

**Skip-gram:** learns to predict the context words from a given word, in the case where two words (one appearing infrequently and the other more frequently) are placed side-by-side, both will have the same treatment when it comes to minimising loss since each word will be treated as both the target word and context word. Comparing that to CBOW, the infrequent word will only be part of a collection of context words used to predict the target word. Therefore, the model will assign the infrequent word a low probability.

Google, Wikipedia and others have made available pre-trained networks that will allow you to take advantage of their models. If you go here you can learn how to build your own predictive word vectors:

<https://towardsdatascience.com/an-implementation-guide-to-word2vec-using-numpy-and-google-sheets-13445eebd281>

We will now demo show how pre-trained vectors can be used in the Python code here:

<https://drive.google.com/open?id=1xhLEXXx-JwcOOtk5qWf2MdeNDXCA4xTT>

## 2.10 Sentiment analysis and opinion mining I

In this step, I will outline two of the most popular concepts in natural language processing. If you use text-based data the likelihood is you will come across both Sentiment and Opinion mining in your research. Liu (2012) gives a description of them as follows:

Sentiment analysis and opinion mining is the field of study that analyzes people’s opinions, sentiments, evaluations, attitudes, and emotions from written language. It is one of the most active research areas in natural language processing and is also widely studied in data mining, Web mining, and text mining. In fact, this research has spread outside of computer science to the management sciences and social sciences due to its importance to business and society as a whole. The growing importance of sentiment analysis coincides with the growth of social media such as reviews, forum discussions, blogs, micro-blogs, Twitter, and social networks. For the first time in human history, we now have a huge volume of opinionated data recorded in digital form for analysis.

Sentiment analysis systems are being applied in almost every business and social domain because opinions are central to almost all human activities and are key influencers of our behaviours. Our beliefs and perceptions of reality, and the choices we make, are largely conditioned on how others see and evaluate the world. For this reason, when we need to make a decision we often seek out the opinions of others. This is true not only for individuals but also for organizations.

The exact difference between the two topics is subtle and practitioners often make it hard to follow.

* An **opinion** can be considered as a person’s ***view*** about something
* A **sentiment** is more of a ***feeling***.

For example, the sentence:

“*I am concerned about the current state of the economy*” 🡺 expresses a **sentiment**

“I think the economy is not doing well” 🡺 expresses an **opinion**.

In a conversation, if someone says the first sentence, we can respond by saying, “I share your sentiment,” but for the second sentence, we would normally say, “I agree/disagree with you.” However, the underlying meanings of the two sentences are related because the sentiment depicted in the first sentence is likely to be a feeling caused by the opinion in the second sentence. We can also say that the first sentiment sentence implies a negative opinion about the economy, which is what the second sentence is saying. Although in most cases opinions imply positive or negative sentiments (Saberi and Saad, 2017).

Before we buy something, we tend to get opinions from different people such as friends or relatives. But now, the first thing we do before buying something is “Google it”. We may use questions like:

* Which car should I buy?
* What schools should I apply?
* Which professor to work for?
* Who should I vote for?

What do we understand from all of this? ***People’s opinions matter!***

“Googling” takes us to millions of blogs and discussion forums. One could easily spend years going through it. We can go through each site one by one and make our own opinions, but here are the questions we need to ask ourselves.

* Is this reliable?
* Can we spot the difference between good and bad reviews?
* How do we know it’s not a fake or paid review?

Searching for reviews is a difficult task. There is no single proper answer we can get for a search like “iPhone vs Samsung phone”. We have to go through each and every review which could be tedious. Additionally, people express opinions in different ways.

If we choose to stick to one website, how we can ensure that the reviews aren’t biased (in that all people might have the same opinion) or fake (such as reviews posted by the people from the company).

Let’s take an example of a review section from Yellow Pages which looks suspicious, why?

* They only have 5 stars reviews.
* All of the dates are close to one another.
* Reviewers seem to know the people from the company by name.

Is this a coincidence or fake, and what can we do about this?

### Subjective Analysis

Subjective Analysis is the computational study of effect, opinions and sentiments expressed in text. It is based on personal opinions, the point of view of a subject. Whereas objective analysis is based on facts, that are measurable and observable.

### Individual perspective

From an individual perspective, people tend to see the pros and cons of a product or service. Online sources, such as reviews and blogs, tend to merge these together in their discussions. The reviews can be complementing or contrasting between different sites, for example:

* **CNET:** *The iPhone lacks some basic features.*
* **Tech Blog:** *The iPhone has complete set of features.*

### Business Perspective

The perspective varies from the point of view of the consortium: owner or competitor. In the case of iPhone review, Apple would be interested in:

* Do they like it?
* What do they dislike?
* What are the major complaints?
* What features should we add?

Meanwhile, Apple’s competitors would be interested in:

* What are iPhone’s weaknesses?
* How do we compete with them?
* Do people like everything about it?

These opinions are valued as business intelligence, which could help product designers to see how well the product is performing in the market and how it can be improved.

### Applications

Subjective analysis is applicable to areas such as:

* Product and Service Benchmarking
* Market Intelligence
* Topic Survey
* Product Reviews
* Tracking political topics
* Place an ad when someone praises the product
* Place an ad from the competitor when someone criticizes the product

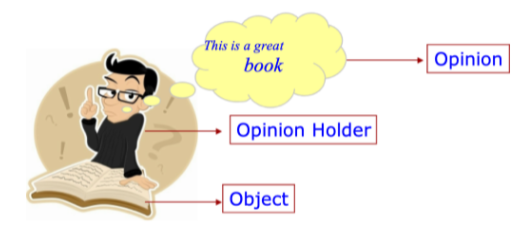
Continue to the next step to conclude our explanation of sentiment analysis and opinion mining.

## 2.11 Sentiment analysis and opinion mining II

### Opinion Mining

Opinion mining is the approach of understanding the drivers behind why people feel the way they do. An opinion consists of three components:

* **Opinion holder:** The person or organization that holds a specific opinion on a particular object.
* **Object:** Item on which an opinion is expressed.
* **Opinion:** A view, appraisal or attitude on an object from an opinion holder.



There are two types of evaluation opinions:

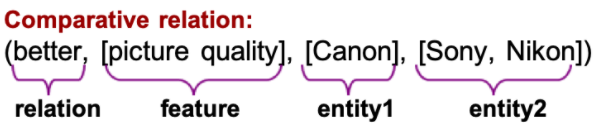
1. **Direct opinions:** This is a direct opinion given by an individual. It is easier to work with this type of opinion. Example: “***This car gives poor mileage.***”
2. **Comparisons:** An opinion given in comparison with another object. This type of opinion requires more insight. Example: “***The Toyota Corolla is not as good as Honda Civic.***”

### Opinion mining sub-fields

Opinion mining has six sub-fields. Let’s look at them briefly below:

1. **Sentiment classification:** Classification of a document/sentence/feature based on sentiments expressed by authors, e.g. ***Positive***, ***Negative***, or ***Neutral***.

2. **Comparative mining:** Identify comparative sentences and extract relations between entities in a sentence. Example: “***Canon’s picture quality is better than that of Sony and Nixon.***”



3. **Opinion integration:** Integration of opinions from different sources such as blog, sites and forums.

4. **Opinion spam/trustworthiness:** Determine likelihood of spam in opinion and also determine the authority of opinion. Characteristics of misleading opinions include repetition in reviews, misleading positive opinion and high concentration of certain words.

5. **Opinion retrieval:** The process **retrieves opinions and ranks them about the topic**. This is similar to Document Retrieval. A relevant document **must satisfy criteria: relevant to the query topic and opinions about the topic**.

6. **Opinion question answering:** This is a process which is applicable to virtual assistants and similar to Opinion Retrieval, where the opinions are retrieved and summarized in a natural language form. Take the example below:

Q: *What is the international reaction of reelecting Robert Mugabe as President of Zimbabwe?*

A: *African observers generally approved of his victory, while western governments strongly denounced it.*

### Sentiment classification

Sentiment classification is the process of classifying documents/sentences/features based on the overall sentiments expressed by authors. These sentiments can be positive, negative, or neutral. It is similar to topic-based Classification, wherein topic words are important, but in sentiment classification, sentiment words, such as great, excellent, or horrible, are important.

Sentiment Classification varies based on the type of input. The different types of input are:

* Sentence
* Document
* Feature

### Sentence level classification

In sentence-level classification, a sentence might have one or multiple opinions. There are two tasks which needs to be accomplished to classify the sentence:

**Task 1**: **Identify if sentence is opinionated.**

The sentence needs to be classifed as objectively or subjectively opinionated.

**Task 2: Determine polarity of the sentence.**

The polarity of a sentence can be defined as the overall sentiment of the sentence, such as *positive*, *negative*, or *neutral*.

The polarity of a complete sentence is calculated by:

**Sum of polarity of all the words in a sentence divided by the total number of words in the sentence.**

### Document level classification

A document can be anything, such as a post or review of a product. It can contain opinions of single or multiple objects, produced by single or multiple opinion holders. The task in this type of classification is to determine the overall sentiment orientation of the document.

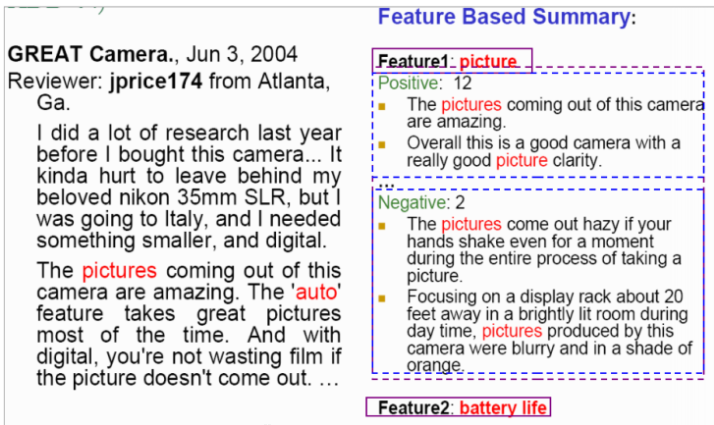
### Feature level classification

Feature level classification results in aggregated opinions made by an opinion holder. The tasks involved are:

**Task 1:** Identify and extract object features that have been commented on by an opinion holder, e.g. “picture” or “battery life”.

**Task 2:** Determine polarity of opinions on features.

**Task 3:** Group feature synonyms.



Go to the following Google Colab to complete this step:

<https://colab.research.google.com/drive/1aW2K5bd8uwMsJqzFwsbpeI45ObSbt1Qf>

# Quiz

**Question 1 –** “San Francisco (is a 2-gram)”

* True
* False

**Question 2 –** N-grams can’t be used to predict the next word.

* True
* False

*Yes, they can. The Python Textblob package as an option that will help make a prediction based on a given n-gram.*

**Question 3** – Calculate the term frequency for the word “the” in the following sentence:

“The car is driven on the road.”

Ignore Case sensitivity and select an answer below.

* 2/7
* 1/7
* 3/7

**Question 4 –** The intuition behind Inverse Document Frequency (IDF) is that a word is very useful to us if it appears in all the documents.

* True
* False

**Question 5 –** Calculate the TF-IDF for the words ‘truck’ and ‘car’ for the following 2 documents:

Sentence 1: The car is driven on the road.

Sentence 2: The truck is driven on the highway.

* & 0.043
* 0.043 & 0.043
* 0 & 0

**Question 6 –** The Bag-of-words model is mainly used as a tool for feature generation.

* True
* False

**Question 7 –** Identify the frequency-based embedding techniques:

* Count vector
* CBOW
* Skip-Gram
* Co-occurrence vector

**Question 8 –** CBOW model predicts the current word given context words within a specific window.

* True
* False

**Question 9 –** You can convert an entire sentence into a one-word vector.

* True
* False

*Each word can be transcribed into a word vector and these vectors can then be added to convert all the text into a single word vector.*

**Question 10 –** Subjective Analysis is the computational study of sentiments expressed in text.

* True
* False

*Subjective Analysis is the computational study of effect, opinions and sentiments expressed in text.*

Topic 3: Introduction to Image Processing

# What is an image?

## 3.2 What is an image?

With modern data analytics problems, we are faced with the challenge of having to incorporate multiple data types in our analysis.

This is predominantly but not exclusively to do with the vast amounts of images and text that are collected for example by social media platforms or autonomous vechicles (have a look at this video, it is interesting):

<https://www.youtube.com/watch?v=tiwVMrTLUWg>

It makes sense to use this data as it holds substantial volumes of information. There are now techniques to help us pre-process and process this data.

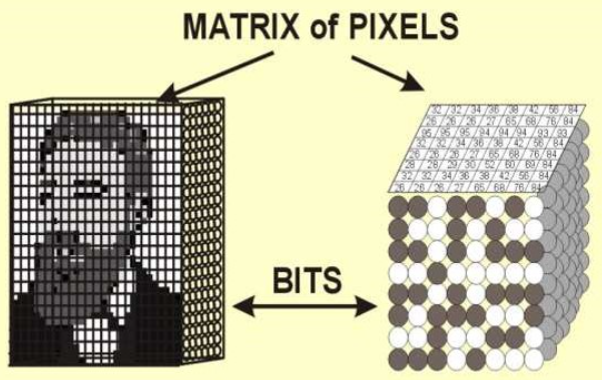
In this topic, we are going to show you a number of techniques that can be used to pre-process images and convert them into a form that can be used by machine learning techniques, such as neural networks. In fact, image processing has become so advanced that we have arrived at the point where AI’s can detect objects and make decisions based on a changing environment.

In this step, we are going to explain how a computer views and interprets an image. Following this step, we will address the following issues:

* The origins of Digital Image Processing (DIP)
* What is DIP?
* Examples of fields that use DIP
* Fundamental Steps in DIP
* Components of an image processing system

### What is an Image?

On a computer, images are stored as pixels. Effectively, they are big matrices where each cell of the matrix represents a colour.



The representation of colours is called Bit Plane. Each bit in the image pixel doubles the number of colours. For example:

* 1-bit gives us two colours.
* 2-bit gives us four colours.
* 4-bit gives us eight colours.

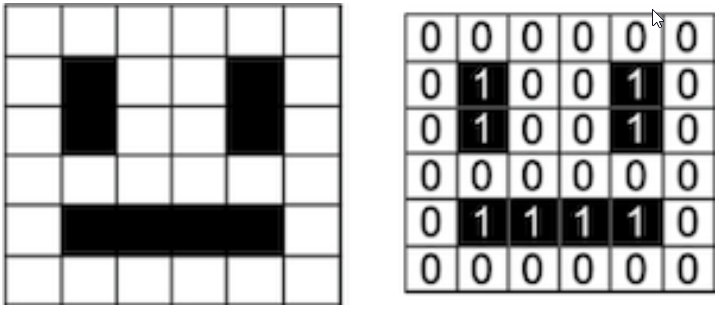
For computer interpretation of images, metadata is also required such as:

* **Colour depth:** how many bits represent each pixel.
* **Resolution:** width & height of image.

### Monochrome (1-bit) Image

A Monochrome, or 1-bit, image requires only one bit to represent each pixel. i.e.

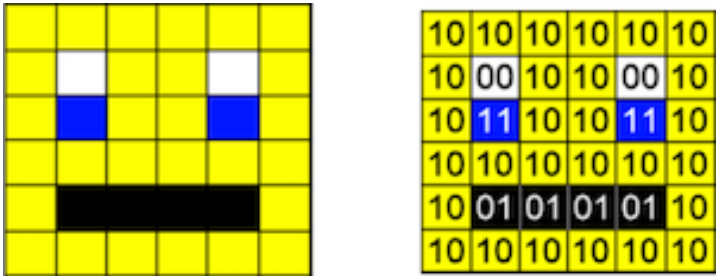
* 0 = white
* 1 = black



### 2-bit image

A colour image below uses 2-bit in each pixel for four colours. e.g.

* 00 = white
* 01 = black
* 10 = yellow
* 11 = blue



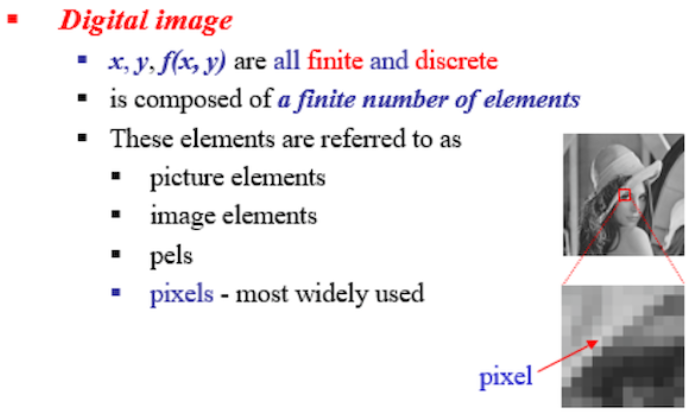
### Analogue Images

Analogue images can be described as basic analogue signals which are manipulated by electrical signals. Examples are television, medical images, etc. Much information cannot be derived from digital images containing only numbers. Hence, digital images produced through various methods, such as medical image processing, need to be converted to analogue images for viewing.

### Digital Images

Digital images are numeric representation, normally binary of a two-dimensional image. It is a matrix of many small elements also called pixels. Each pixel is a numeric value. These pixels relate to the brightness or colour, which we see with our naked eye when converted from digital to analogue. Digital images are used for different purposes to be:

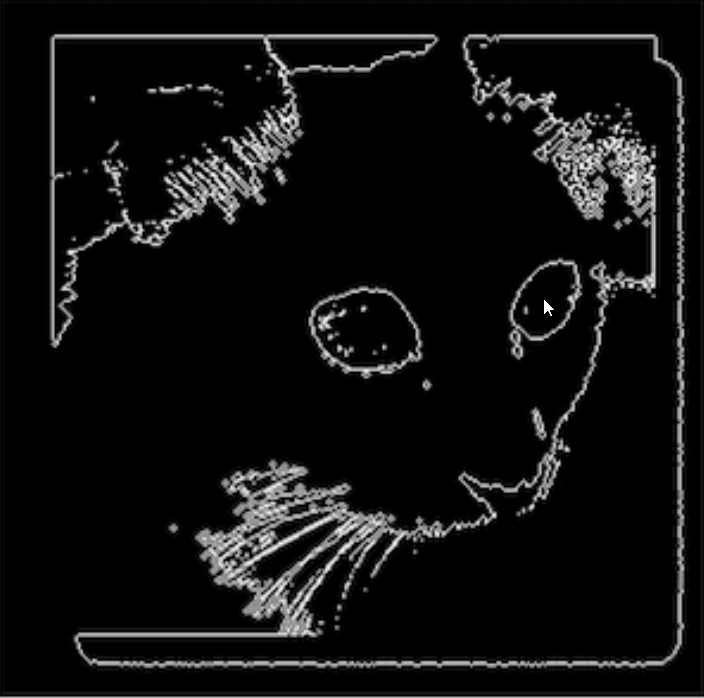
* Viewed by a human for pleasure or aesthetics.
* Viewed by humans for interpretation.
* Viewed by machine in order to retrieve information about its surroundings.
* Used by machine to extract information and feed it to another machine or to be displayed to human beings.



Digital images can be classified into four types:

1. **Binary/Monochrome image:** This is the basic black and white image consisting of 1bit/pixel i.e value range:

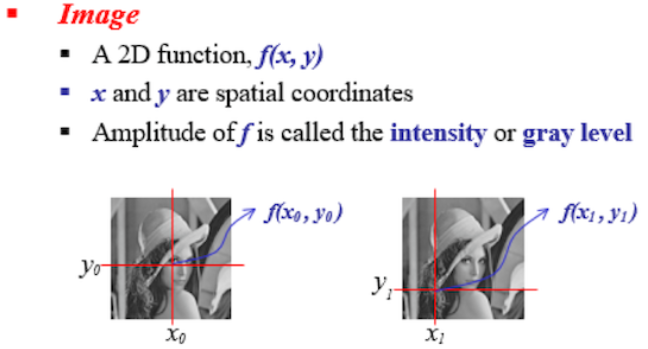
* 0 = white
* 1 = black



2. **Colour Image:** Colour images consist of 24bit/pixels, i.e 3 bytes/pixel. It consists of Red, Green and Blue colours in the range:

R=[0,255], G=[0,255], B=[0,255].

3. **Greyscale Image:** A grayscale image may be defined as two dimensional light intensity function ***f(x, y)***, where ***x*** and ***y*** denote spatial coordinates and the amplitude or value of at any point ***(x, y)*** is called intensity, greyscale, or brightness of the image at that point.



A greyscale image is one in which the value of each pixel is a single sample representing only an amount of light, that is, it carries only intensity information. Greyscale images are a kind of black-and-white or grey monochrome and are composed exclusively of shades of grey. The contrast ranges from:

* black 🡺 **weakest** intensity, to
* white 🡺 **strongest** intensity

Greyscale images are distinct from one-bit bi-tonal black-and-white images which, in the context of computer imaging, are images with only two colours: black and white (also called bi-level or binary images). Grayscale images have many shades of grey in between.

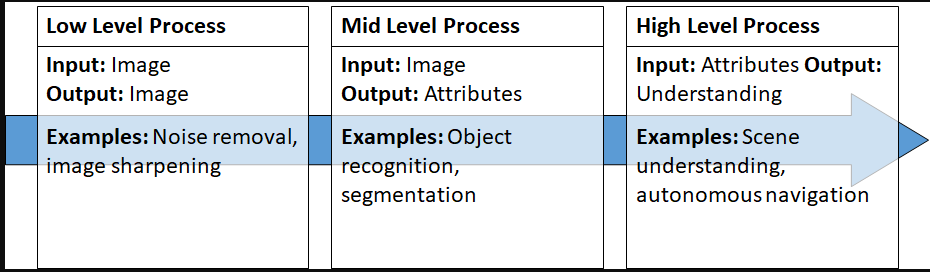
# Image processing

## 3.3 What is digital image processing?

Glen MacDonald from the University of Washington said that Digital Image Processing is “***defined as a means of translation between the human visual system and digital imaging devices. The human visual system does not perceive the world in the same manner as digital detectors, with display devices imposing additional noise and bandwidth restrictions***.”

In many vision-based machine learning algorithms, we may have to pre-process the image before we submit it to a black-box algorithm. So, for example, imagine that you have thousands of pictures each with differing sizes and lighting quality. You may want to identify if there are cats in the pictures. The problem you will have is that you are going to have to ***standardise*** your input images in order for your classifier to work properly.

You will find that it can be hard to differentiate between Image Processing and Image Pre-Processing. I tend to think of Pre-Processing as a more premitave processing stage, but this is not a strict rule. Generally, Image Processing in Computer Vision problems can be broken up into low-, mid- and high-level processes, such as those outlined in Figure 1.



### Why do we do it?

Pre-processing is not a certain technique and can be anything you do on your images before using them. For example, applying filter, translation, rotation, or binarization. Even image cropping can be considered as pre-processing. It is highly dependent on what you are doing and the application itself and can contain a number of functions on each of your images.

Digital images can be used for different purposes to be:

* Viewed by human beings for pleasure or aesthetics.
* Viewed by human beings for interpretation.
* Used by a machine in order to retrieve information about its surroundings.
* Used by a machine to extract information to be fed to another machine or to be presented to a human being.

For the means of simplicity, we define pre-processing as a common name for operations with images at the lowest level of abstraction, both the input and output are intensity images. The aim of pre-processing is an improvement of the image data that suppresses unwanted distortions or enhances some image features important for further processing.

### Image pre-processing vs image processing enhancement

Image enhancement is made up of a set of algorithms aiming to fit an image to the human visual system. But most of the time, the images do not fit those criteria. This can be because the geometry is distorted, the lighting is irregular, too strong or insufficient, there is noise in the images. This is where pre-processing comes into play. **Pre-Processing algorithms aim to prepare data, i.e. images, so they can be used efficiently by other types of algorithms, for example, general image processing, image enhancement, or image analysis**.

#### Image Pre-Processing

* Input: an image usually coming from a camera or a sensor.
* Output: an image meant to be an input to another algorithm.

#### Image Enhancement

* Input: an image coming from pre-processing.
* Output: an image to be viewed by a human eye.

#### Image Analysis

* Input: an image coming from pre-processing.
* Output: measurements and extracted knowledge, usually not stored as an image, can be used by a human being or it can be fed to an expert system or a conventional algorithm to populate a database, or make decisions in an embedded system.

In the next step, we will look at some image processing tools that can help us enhance or standardize our images. Can you think of any useful “functions” that would help in our Image pre-processing?

## 3.4 Digital image processing cycle

In the previous step, we showed you how to implement a number of useful image processing tools

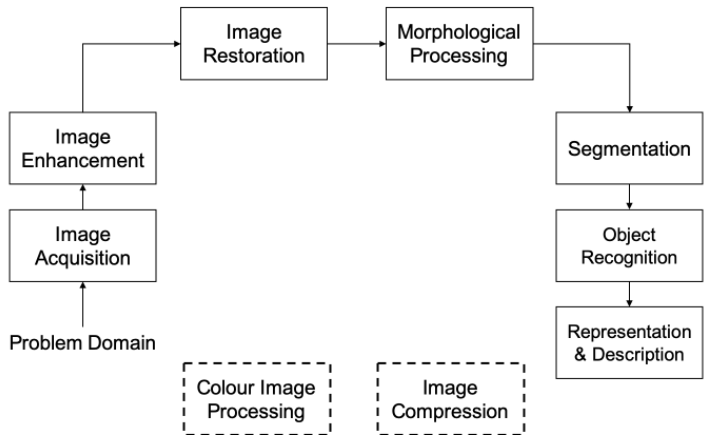
The continuum from Image processing to computer vision can be broken up into low-, mid- and high-level processes:

1. **Low-level Process:** The first step involves putting the image through various methods such a noise removal, sharpening, filter, etc., to prepare for further processing.
2. **Mid-level Process:** This process involves methods such as object recognition, segmentation, etc., to extract the attributes.
3. **High-level Process:** This process involves methods such as scene understanding, autonomous navigation, etc., resulting in the description of the image.

Digital Image processing can be described as performing digital signal operations on digital images. Major interest in digital image processing stems from two application areas:

1. Improvement of pictorial information for human interpretation.
2. Processing of image data for storage, transmission and representation of autonomous machine perception.

Digital image processing is performed in nine key stages. These key stages can be categorised into the three processes mentioned above.



We are going to predominantly look at low-level processing techniques, however, some of these can be used in mid-level processing.

### Image acquisition

The image can be available from various sources such as camera, video etc. The input may have to be digitized if the source is analogue, using an analogue to digital converter.

### Image pre-processing

Pre-processing is the first step in image processing. The source of our image can be from various sources such as a camera or a sensor. These images might not be fit for further processing. For example, distorted geometry, blurred image, irregular lighting and noise. We can address these issues by pre-processing the images. Some of the steps are:

* Filtering
* Image Cropping
* Rotation
* Standardisation
* Colour Transforms

The pre-processing of an image is done according to the requirements of the application and the input required for the next step such as image enhancement and image analysis. It may be some of the above-mentioned steps or all of the steps.

## 3.5 Digital image processing tools

Digital image processing can be described as performing digital signal operations on digital images. Major interest in Digital Image processing stems from two application areas:

1. Improvement of pictorial information for human interpretation.
2. Processing of image data for storage, transmission and representation of autonomous machine perception.

In this step, we will outline a number of techniques that can be used in image pre-processing and image enhancement.

Image pre-processing refers to operations done before a key processing step, such as:

* Filtering
* Colour transform
* Sub-sampling/scaling
* Histogram generation

Image enhancement refers to operations to improve the look of an image, such as:

* Contrast Enhancement
* Histogram Equalization
* Filtering
* Sharpening

We will implement some Python code in the following steps to demonstrate a number of the processing techniques. These techniques can be used in either pre-processing or image enhancement and will cover the following pre-processing techniques, which are as follows:

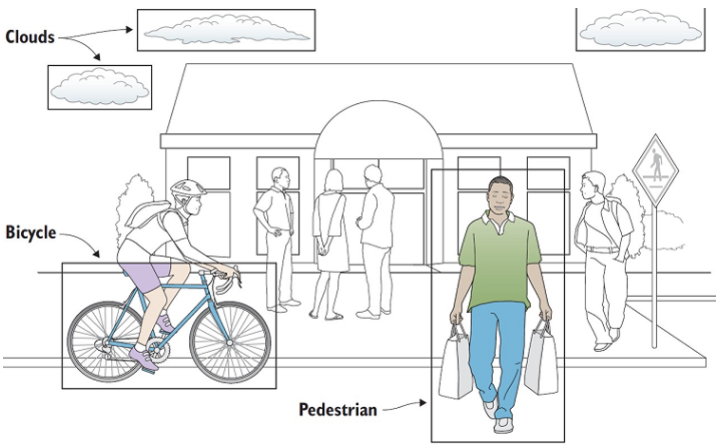
* Color transformation
* Standardization
* Thresholding
* Data augmentation(such as translation, rotation, shearing, horizontal or vertical flips)
* Image smoothing
* Image gradients
* Canny edge detection
* Morphological transformations
* Image sharpening

## 3.6 Colour transformation

This step will consider methods of colour transformation.

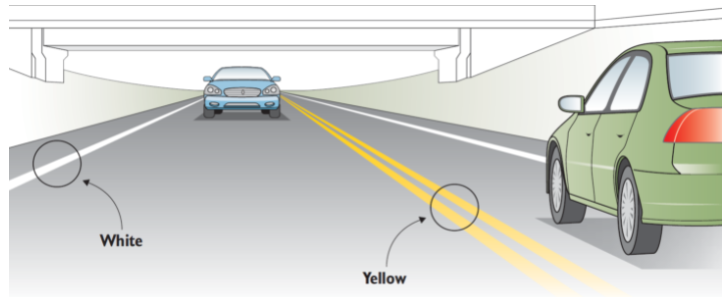
The first method we will look at is a colour transform. Colour transform converts colour images to greyscale in order to reduce computation complexity. This can help machine learning algorithms focus more on the objects rather than on the colour. It also reduces the dimensionality of the data, as colour images have a m×n×3 matrix as opposed to a m×n matrix for grayscale images. The choice of greyscale over colour can be algorithm-specific.

You can see from Figure 1 how patterns in brightness and darkness of an object (intensity) can be used to define the shape and characteristics of many objects.



In Figure 1 above, colour may not be necessary as we are really interested in the shape. However, in other applications, colour is important as it is used to define certain objects.

Whereas, in Figure 2 below, you can see the colour of the road markings is extremely important or in the case of skin cancer detection where medics rely heavily on the skin colours (red rashes).



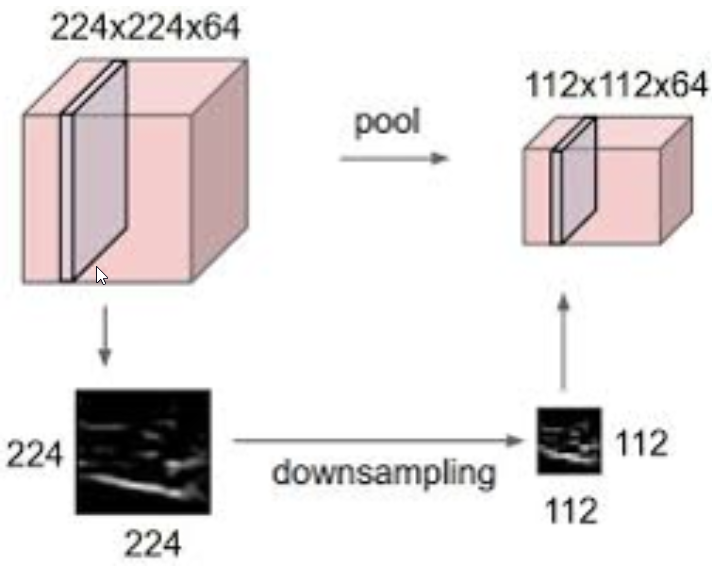
Complete this step by following the link Google Colab here:

<https://drive.google.com/open?id=1y7BClILQGsimH1MgBEgaYEERQrzuTsM0>

## 3.7 Standardisation

One important constraint that exists in some machine learning algorithms, such as Convolutional Neural Networks (CNN), see Figure 1, is **the need to resize the images in your dataset to a unified dimension**.

This implies that our images must be pre-processed and scaled to have identical widths and heights before fed to the learning algorithm.



In the following code, we resized the img\_color to 200x100. The following Google Colab notebook is quite useful in helping to understand the functionality of OpenCV.

Go to the following Google Colab to complete this step:

<https://drive.google.com/open?id=1zvntj2TlozgFC1xsEBDoajN8lgX8_W6A>

## 3.8 Image thresholding

Thresholding is the process of assigning either of two values to pixels. The simplest form is known as simple thresholding, where we assign a white- or black-based pixel, based on a comparison of the original pixel value to a threshold value.

For example, if a pixel value is greater than a threshold value, it is assigned one value, such as white, or else, it is assigned another value. This can also be done in colour as we can see from this image.



There are many different forms of thresholding. They can be generally categorized into the following groups:

* **Simple threshold:** If the pixel value is greater than a threshold value, it is assigned one value, e.g. white, else it is assigned another value, e.g. black. There are a number of options for this type of Thresholding.
* **Adaptive thresholding:** In simple thresholding, a global value is used as threshold value. This may not work in some circumstances where an image has different lighting conditions in different areas. In that case, Adaptive thresholding can be very useful, as **the threshold varies for a small region of the image**.
* **Otsu’s binarization:** Most images can be represented by a histogram going from 0 to 255. If you convert an image to greyscale and then plot the histogram of the pixel counts, you will see in some cases, a bimodal distribution or two peak distribution. Otsu’s binarization attempts to determine the minimum point between the two peaks, and this is then used as the separating factor.

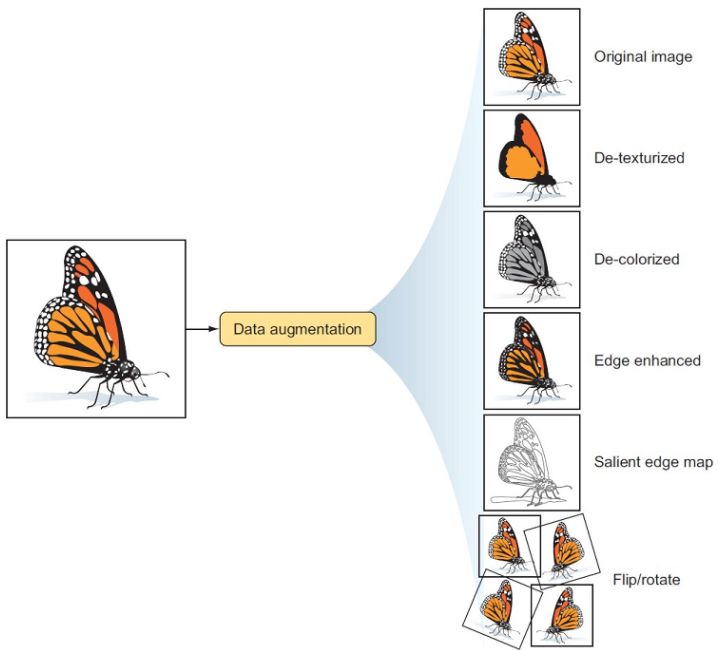
To complete this step go to this Google Colab:

<https://drive.google.com/open?id=1X4Rs7sseU7KM_wz2XeW1ocw7PRj2AjH4>

## 3.9 Image data augmentation

Data augmentation is a common pre-processing technique, which involves the augmentation of the existing dataset with perturbed versions of the existing images.

Scaling, rotations and other affine transformations are typical. This is done to **enlarge your dataset and expose the neural network to a wide variety of variations of your images**. By doing this, it is more likely that your model recognizes objects when they appear in any form and shape. Here’s an example of image augmentation applied to a butterfly image:



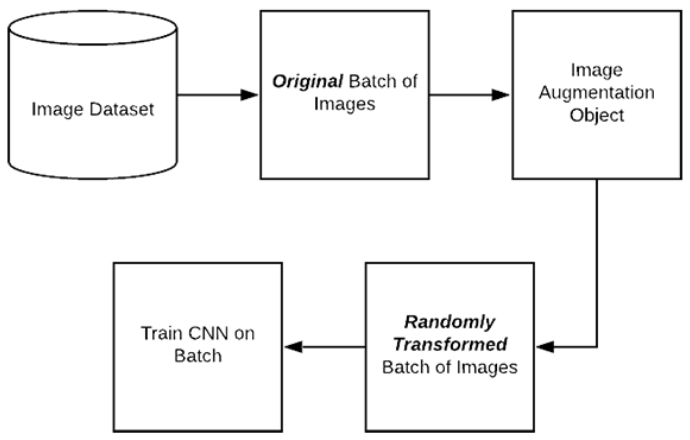
In this Step, we will examine a number of tools that OpenCV gives us to augment our image datasets. We will give a brief explanation of each image transformation technique and implement them using Python code. But first, we will look at a common data augmentation process outlined by Keras:

<https://keras.io/api/preprocessing/image/>

The diagram in Figure 2 which is outlined by PyImageSearch:

<https://pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/>

gives a nice visualisation of a data augmentation on the fly. On the fly means we will augment the data as we train a machine learning algorithm, and we will only use the augmented data.



PyImageSearch:

<https://pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/>

outlines the steps involved as follows:

1. An input batch of images is presented to the ImageDataGenerator.
2. The ImageDataGenerator transforms each image in the batch by a series of random translations, rotations, etc.
3. The randomly transformed batch is then returned to the calling function.

Please be aware that ImageDataGenerator only returns the generated image. Each image is transformed and then sent to the training phase. So, the training data is completely built using generated images.

Now, let’s look at some augmentation techniques:

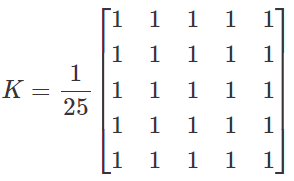
<https://drive.google.com/open?id=1F24ZIPwUi4fYJ0y2vHMdpDjv66fA6xeZ>

## Image smoothing

**Image smoothing is done to remove noise from an image. It reduces the sharpness of edges and smoothness the details in the image.**

The idea is to highlight important features and reduce the effect of noise. Image smoothing is also known as Blurring, OpenCV, and is achieved by convolving the image with a low-pass filter kernel.

Convolution provides a way of multiplying together two arrays of numbers, generally of different sizes. You can think of it as sliding the smaller matrix across the bigger matrix to produce a matrix of the same size as the original matrix. A kernel is usually a matrix/filter that is convoluted with an image. It removes high-frequency content, e.g., noise or edges from the image resulting in edges being blurred when this filter is applied. Matrix K below is a 5 \* 5 averaging convolution kernel.



There are a number of techniques that can help achieve this and they are as follows:

* Averaging
* Gaussian filtering
* Median filtering
* Bilateral filtering
* Denoising (not theoritically a Smoothing technique but is close)

We will now examine each filter in the following Google Colab:

<https://drive.google.com/open?id=1EBzXvfhmDR79d89te_bT1zurVBWCkjSj>

## 3.11 Morphological transformations

**Morphology is a broad set of image processing operations that process images based on shape**s.

It applies a structuring element to the input image and gives an output of the same size. In this operation, each pixel in the output is the result of a comparison of input image pixels with its neighbourhood pixels. The different types of operations are Dilation and Erosion.

Please go to this Google Colab to complete this step:

<https://drive.google.com/open?id=1XzMS7e0iG_98DqtuK6kyuIi4zZOqzdQr>

## 3.12 Sharpening an Image

In this step, we will examine a number of techniques that will **highlight edges or make them more prominent**. The areas we will examine are:

* Canny Edge Detection to detect edges
* Image Gradients to make edges prominent in images
* Sharpening an Image to increase the contrast between bright and dark regions

### Canny Edge Detection

Canny Edge Detection, OpenCV, is a popular algorithm to detect edges in images other than Image Gradients. The algorithm is applied in multiple stages such as:

**Noise Reduction:** Edge detecion is susceptible to noise. Hence, the first step is to remove the noise from image with Gaussian filter.

**Find gradient intensity:** The smoothened image is then filtered with a **Sobel kernel** in both the horizontal and vertical direction to get first derivative in the horizontal direction ***GxGx*** and the vertical direction ***GyGy***. Sobel derivative is applied to the image to make the edges prominent. From these two images, we can find edge gradient and direction for each pixel as follows:



**Non-Maximum Suppression:** After finding the intensity and direction, unwanted pixels are removed from the image by **checking the local minimum in the direction of gradient. If negative, the pixel is suppressed to zero**.

**Hysteresis Thresholding:** Hysteresis Thresholding is a Thresholding process to differentiate between edge and non-edge pixels. Here, we consider two values: *minVal* and *maxVal*. All pixels above *maxVal* are considered edges and all below *minVal* are categorised as non-edges and discarded. The values lying between *minVal* and *maxVal* are considered as edges based on the connection with the local minimum.

Now, import the image in Figure 4 here and store in the folder that is currently mounted:

<https://www.computing.dcu.ie/~amccarren/mcm_images/noisy_image2.png>

We are going to run canny edge detection from OpenCV. You will see it works pretty well. Try it with the median blur filter from OpenCV.

Follow this link to go to the Google Colab for this step:

<https://drive.google.com/open?id=1pvOLCtqTw5IzbhOqOXMCjCt71wndz9SS>

# Quiz

**Question 1 –** A monochrome image requires one bit to represent each pixel.

* True
* False

**Question 2 –** Colour images consist of 16 bits/pixel.

* True
* False

*Colour images consist of 24bit/pixels, i.e 3 bytes/pixel. It consists of Red, Green and Blue.*

**Question 3 –** Object detection can be considered to be a low-level image process.

* True
* False

*Low-level process are processes that get an image for further processing such as object detection. Object detection is a mid-level process.*

**Question 4 –** We should always use colour images when applying machine learning techniques.

* True
* False

**Question 5 –** When eroding an image we slide a kernel across the image and a foreground pixel value is considered 1 only if all the pixel values in the kernel window are 1 else the pixels values are eroded to 0.

* True
* False

**Question 6 –** We sometimes augment images by perturbing the existing versions of the image. This can be very useful in identifying edges on images.

* True
* False

*Augmented images are usually used to enlarge a dataset and attempt to expose the learning algorithm to a wide range of variation.*

**Question 7 –** Which filtering technique best preserves the edges on an image?

* Median filtering
* Gausian Filtering
* Bilateral Filtering

**Question 8 –** Which Image Thresholding technique requires you to specify the threshold of the pixel intensity?

* Adaptive
* Otsu’s Binarization
* Simple Thresholding

**Question 9 –** Which images if any are the result of a median blur on the image below:



Both images have had a median blur. The first shows the result of a median blur and the second is the canny edge detection of a median blurred image.

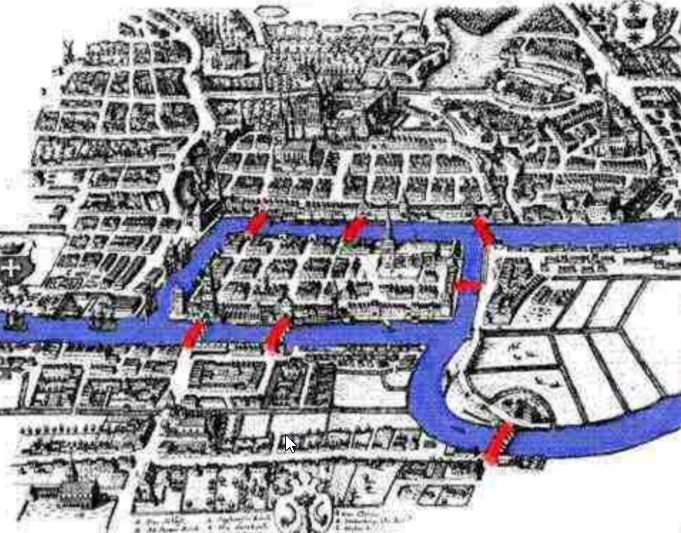
Topic 4: Introduction to Graph Data

# Topic 4: Introduction to Graph Data

# Introduction to graph data

## 4.2 Intro to graph data

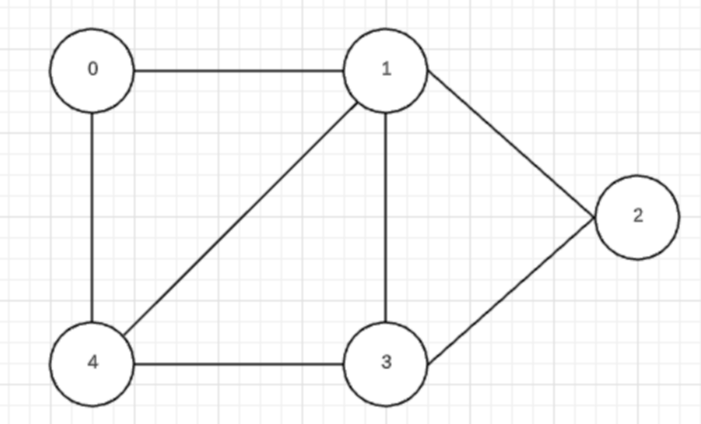
In the 18th century, a Swiss mathematician called Leonhard Euler proposed the basic idea of graphs. His eventual solution to the famous Königsberg bridge problem depicted below is commonly referred to as the origin of graph theory:



<https://towardsdatascience.com/graph-theory-history-overview-f89a3efc0478>

### What is a graph?

A mathematical graph is the study of the relationship between objects and can be described as a set of nodes that are connected together. Figure 2 shows a connected graph. The objects are referred to as **nodes** (or vertices), and they are **connected by edges** (or links).



### Where would we find graphs?

Graphs have many uses inside and out of data analytics. A map of users on a social media network would be a graph. In linguistics, a parse tree displaying the grammatical structure of a sentence would be a graph.

### When would we use graphs?

Graphs are used when more dynamic solutions are required from data. Graphs not only store data about objects but also about the relationships between these objects. For example, Netflix would recommend shows for one particular user by comparing them with users with similar viewing patterns.

### Why do we use graphs?

By examining the relationships between objects, **graphs can find information hidden in the data that might not be obvious when using structured data**.

# What is a graph?

## 4.3 What is a graph?

A mathematical graph can be described as **a set of objects that are connected together**.

Here, the objects are referred to as nodes (or vertices), and they are connected by edges (or links).

#### Facebook example

A typical example would be a social network such as Facebook. On Facebook, everything is a node. **Nodes include:**

* User
* Photo
* Album
* Event
* Group
* Page
* Comment
* Story
* Video
* Link

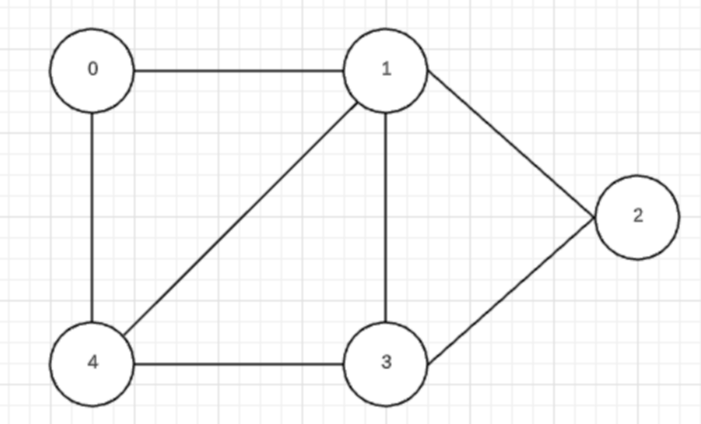
**Anything that has data is a node**!

**Every relationship is an edge from one node to another**. Whether you post a photo, join a group or like a page, a new edge is created for that relationship.

More precisely, a graph is a data structure (V,E) that consists of:

A collection of vertices V.

A collection of edges E, represented as ordered pairs of vertices (U,V).



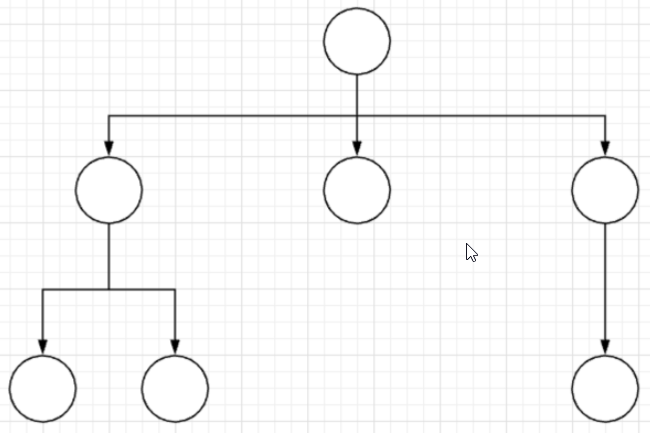
Following this, that means that all of Facebook is a collection of these nodes and edges. This is because Facebook uses a graph data structure to store its data.

### Directed Vs Undirected Graphs

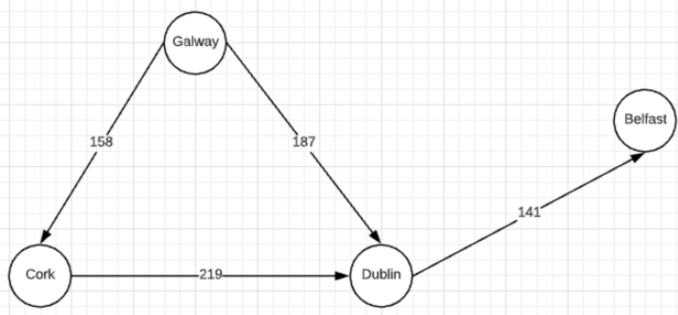
A variation of the graph is whether or not the graph is directed.

**Directed graph** 🡺 *edges only point in one direction*

An example of a directed graph would be a social media graph where one user follows another.



The opposite of this is the **undirected graph**. An example of an undirected graph would be a graph of cities where the edges represent the distance between two cities.



The following are some key terms that are used in relation to graphs:

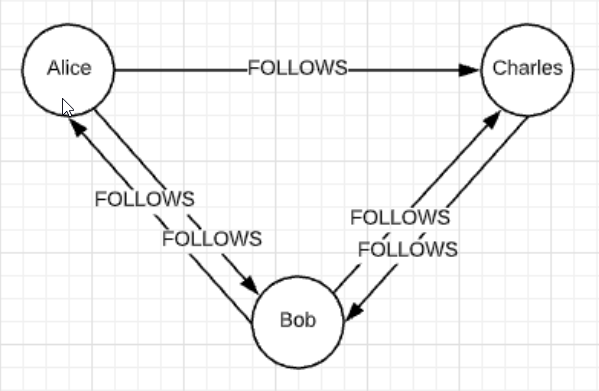
* **Adjacent:** A node is considered adjacent to another if they are connected by an edge. A vertex is said to be adjacent to another vertex if there is an edge connecting them.
* **Path:** A path is a sequence of edges between two nodes. A sequence of edges that allows you to go from vertex A to vertex B is called a path.
* **Degree:** The number of edges connected to a node.
* **Isolated Node:** A node is isolated if it has a degree of 0, i.e. it is connected to no other nodes. Figure 3 shows how Cork has a degree of 2 and the path between Cork and Belfast is either through Dublin or Galway and Dublin.

## 4.4 Where are graphs used?

Outside of data analytics, graphs have many other practical uses.

* Google **Maps** uses graphs in conjunction with GPS to find the shortest path from one destination to another.
* In computer science, graphs are used in conjunction with algorithms to find the **shortest and efficient path** from one node to another.
* In science, graphs can be used to model the **DNA** structure of an organism.
* Operations research uses graphs to find the **optimal route** in order to optimize transportation costs. The travelling salesman problem is a typical example of this.
* In linguistics, graphs can be used to represent the **grammatical** **structure** of a sentence.

Another example that most people would be familiar with would be **social media**. A social media network is quite nicely modelled with a graph. In this example, each node would represent a user, and the directed edges would represent one user following another.



In this step, we will demonstrate how to visualise a graph using Python and how to traverse a graph using an algorithm known as Dijkstra’s algorithm. Follow this link to go to the Google Colab for this step:

<https://colab.research.google.com/drive/1JINhSHQ1AQ-nzgHTPRK8H1qLPrj6lvDu>

## 4.5 Graph vs structured data

Many of you will have come across database systems such as SQL Server or MySQL. These systems are typically structured in a tabular fashion.

Their use boomed in the 1980s when IBM and Oracle developed substantial systems so that businesses and organisations could use them. The problem with the conventional systems was that they were quite restrictive. Structured data is data that has been organized into a formatted repository, typically a database such as SQL Server, MySQL, or just plain old CSV files. Its elements can be made addressable for more effective processing and analysis. In a structured database, each field is discrete and its information can be retrieved either separately or along with data from other fields by joining tables. The power of the database is its ability to make data comprehensive so that it yields useful information.

Over the past 20 years, the use of graph databases has grown substantially. **Graph databases are the result of crossing graph theory with a database. The key innovation here is that the relationship between the data is as important as the data itself**, which is not necessarily the case in systems such as SQL servers. When analysing graphs, you can use the relationships between nodes to infer the organization and dynamics of complex systems. T**his lets you uncover hidden information about the data, test hypotheses, and make predictions about behaviour**.

As opposed to the table structure of traditional databases, graph databases take a different approach:

* Following graph theory, a graph database stores its data in nodes
* Nodes are connected to other nodes via relationships.
* Both nodes and relationships have properties, which are name/value pairs, which tell us more about that particular object.
* They also have labels that differentiate the various types of data stored in a graph database.

**Advantages of graphs over structured database systems:**

* Graph databases are **more dynamic** because they are not limited by dimensional constraints.
* The **performance is constant** since traversing a given vertex’s neighbour nodes via edges are independent of the graph size.
* An **excellent solution** for real-time big data analytics queries where data size grows rapidly.
* The data captured can be easily changed and extended for additional attributes and objects, which gives it **flexibility**.
* It is **index free**.

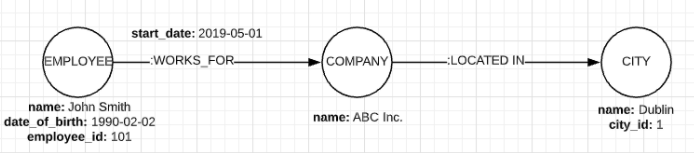
**Disadvantage of graphs over structured database systems:**

Currently, the technology used to query graphs is not as advanced as it is for structured data. If you want a more detailed explanation or are just interested in getting more familiar with them, read the first couple of chapters from this ebook by Xu, et al (2018):

<https://cdn2.hubspot.net/hubfs/4114546/Collateral/Ebook-Native-Parallel-Graphs-The-Next-Generation-of-Graph-Database-for-Real-Time-Deep-Link-Analytics.pdf?__hssc=3506223.1.1581073638892&__hstc=3506223.55a38a4751b03ed9d113394c799da787.1581073638892.1581073638892.1581073638892.1&__hsfp=3480816081&hsCtaTracking=183e3db3-54df-4f24-b6db-b3a3b91aa1d7%7C6bb9bbcf-44eb-4de0-830e-fc9a3979439e>

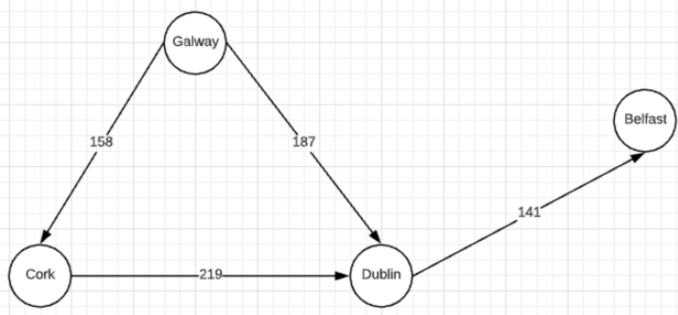
As you can see from the example in Figure 1 below

* **nodes** are modelled on objects and 🡺 thus are ***nouns***
* **relationships** represent actions 🡺 are therefore **verbs**.



**Neo4j is a graph database management system**. It is also what is referred to as a native graph database because it efficiently implements the property graph model down to the storage level.

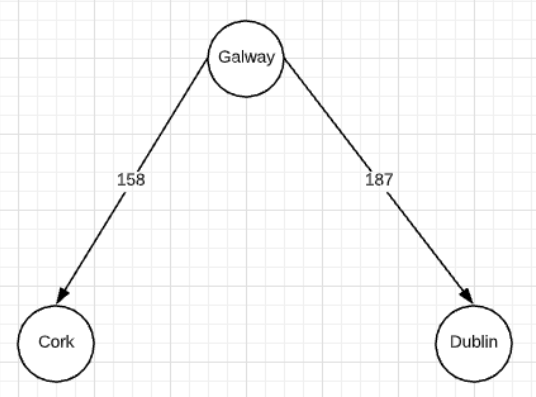
Similar to SQL, graph databases come with their own **query language called CYPHER**. As a query language, **CYPHER focuses on what the result should look like rather than how to achieve the result**. The following is a worked example in CYPHER. Take the cities graph from an earlier section.



If we wanted to only see the cities connected to Galway, we would run the following command:

match (n)-[]->(m) where n.name='Galway' return n,m

Running the above query on the cities graph would produce the following result [Cork, Dublin]:



Have a look at the first two references below. We are not going to delve into Neo4j as it is out of the scope of this course.

<https://neo4j.com/>

<https://neo4j.com/graph-algorithms-book/>

# Converting graph data to a matrix

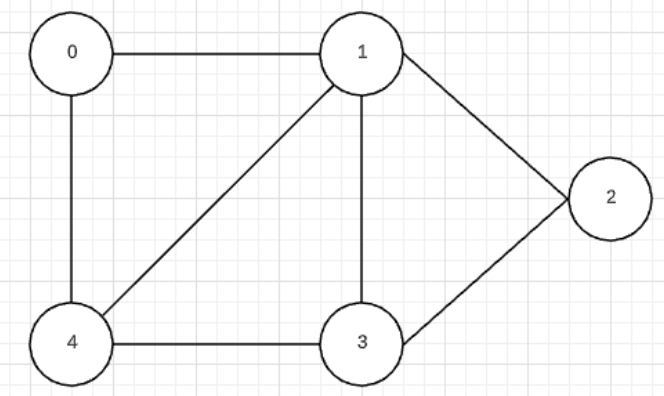
## 4.6 Converting graph data into a matrix

This step considers why would we convert a graph into a matrix.

There is a very clear reason why this conversion takes place. As we shall see later in this topic, having the graph data in a matrix format allows us to perform certain actions on the data that we would not otherwise be able to do. Spectral Clustering, an algorithm we will cover later in this topic, is a key example of this.

**Worked example**

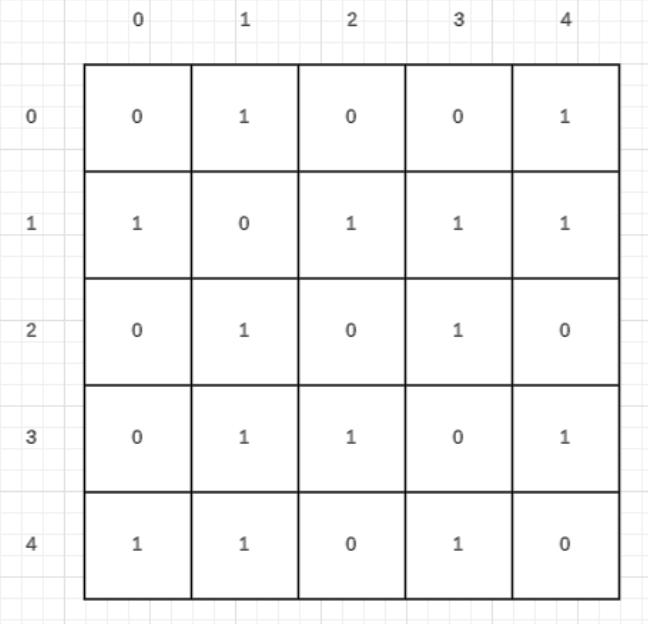
Take this example of an *undirected graph*:



This particular graph has five nodes. So, the first step in turning this graph into a matrix is to create an NxN matrix (or 2D array), where N is the number of nodes in the graph, which in this case is five.

In order to represent the edges in the graph, imagine that ***adj[i][j]*** is the matrix entry that indicates whether there is an edge connecting node ***i*** to node ***j***. If there is indeed an edge connecting these two nodes, then this entry will contain a 1 or the weight of that edge. All other entries will contain 0, if there is no edge between those nodes.

Using the steps above will creating the corresponding adjacency matrix for the above graph:



Something to note is that adjacency matrices for undirected graphs will always be symmetric, as is the case with the above example.

We are now going to use the example from Step 3.4 and converting it to a NumPy matrix. To complete this step please go to the following Google Colab:

<https://drive.google.com/open?id=1wxT8re0njrB7OGJ3K7gF880sUD48Jb50>

## 4.7 Applying Eigenvalues and Eigenvectors to graphs

In this step, we will discuss Eigenvalues and Eigenvectors to graphs

### What are Eigenvalues and Eigenvectors?

* **Eigenvectors** are a special set of vectors associated with a linear system of equations (i.e., a matrix equation) that are sometimes also known as characteristic vectors, proper vectors, or latent vectors (Marcus and Minc 1988, p. 144).
* **Eigenvalues** are a special set of scalars associated with a linear system of equations (i.e., a matrix equation) that are sometimes also known as characteristic roots, characteristic values (Hoffman and Kunze 1971), proper values, or latent roots (Marcus and Minc 1988, p. 144).

The most successful application of them is Google’s page ranking optimisation algorithm. The $25,000,000,000 Eigenvector paper here is a really nice real-life application.

<https://www.rose-hulman.edu/~bryan/googleFinalVersionFixed.pdf>

They are used in many areas including physics, engineering, mathematics and computer science:

* One of the big applications is in PCA which we covered in Step 2.4.9 in the second course in the program
* Pre-processing Data and Feature Impact Calculation.
* They are also used in a different way to identify weak points in network connections.

### What do they tell us?

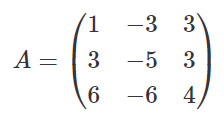
Eigenvalues and eigenvectors allow us to “reduce” a linear operation to separate, simpler problems. Here are some real-world examples of the use of eigenvalues and eigenvectors:

* **Electrical engineering:** Here, they are useful for decoupling three-phase systems through symmetrical component transformation.
* **Oil extraction:** Oil, dirt, and other substances all give rise to linear systems which have different eigenvalues, so eigenvalue analysis can give a good indication of where oil reserves are located.

I often ***think of Eigenvalues as telling us the amount of energy given out from a combination of eigenvectors***. This is related to spectroscopy, but I won’t go there.

### Worked example of calculating Eigenvalues and Eigenvectors for a given matrix

Below is the matrix we will calculate the eigenvalues and eigenvectors for:

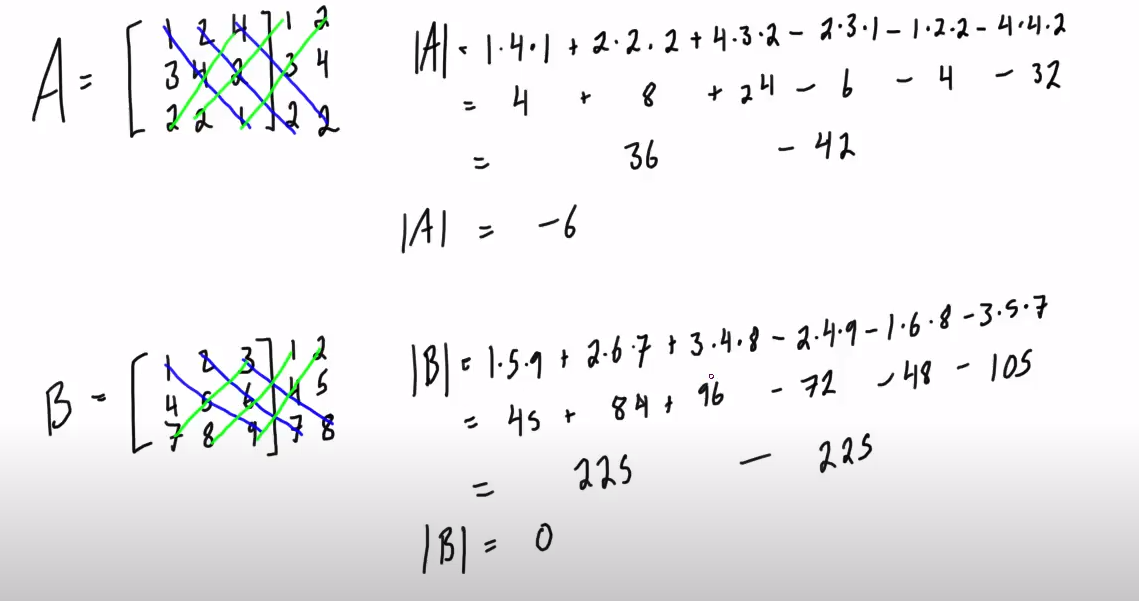


In this particular kind of problem, it is necessary to calculate the eigenvalues first. You can follow the complete solution to this problem here:

<http://wwwf.imperial.ac.uk/metric/metric_public/matrices/eigenvalues_and_eigenvectors/eigenvalues2.html>

Some of you may not remember how to get the determinant of a matrix. This YouTube video should help.

<https://www.youtube.com/watch?v=mEeHxKH46O0>



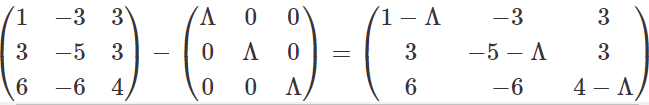
**Calculating the Eigenvalues**

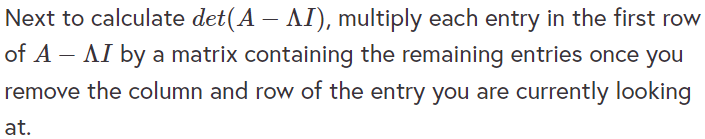
Eigenvalues are the values, , which satisfy the characteristic equation of the matrix, A, which is:

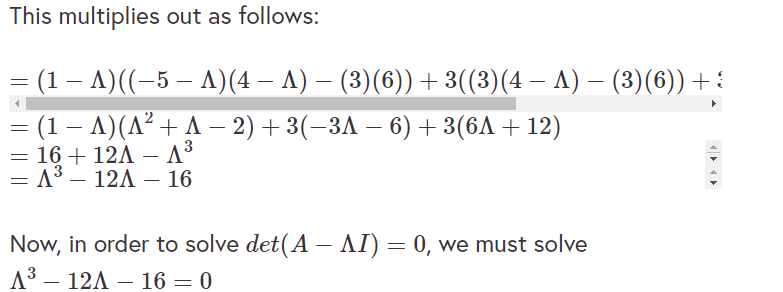


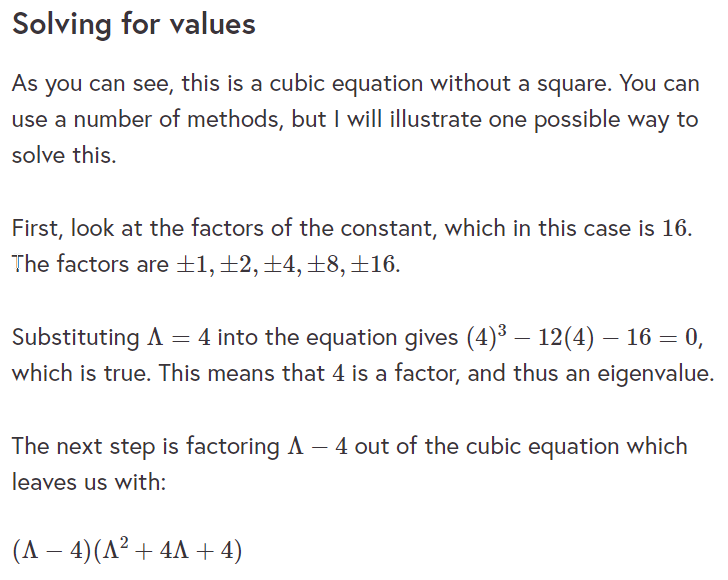
where ***I*** is the matrix is the identity matrix with the same dimensions as the matrix, ***A***.

**Calculating** 

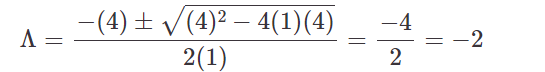








As you can see, this gives us a quadratic equation, which can use to solve for the remaining values using the quadratic formula.



In this case, -2 is a repeating value. So, there is only two roots for this equation.

The two eigenvalues for this matrix are

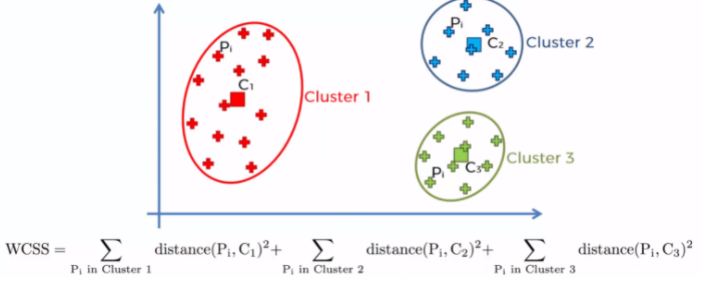
So we have shown how to calculate eigenvalues and eigenvectors. Let use them to show us how to cluster a graph. Follow this link to go to the Google Colab for this step:

<https://drive.google.com/open?id=1-JnWuCa7WHChigtM9n0-3ySmI7yIi1t_>

# K-Means

## 4.8 K-Means recap

In Step 4.9 in Feature Engineering, we covered a clustering algorithm called K-Means. As we pointed out previously, this algorithm is far from perfect and revolves around **minimising the within-cluster variation**. Figure 1 shows how the cost function is calculated relative to the variation in each cluster.



I will not go any further into this at this stage, other than suggesting that you have a look at Step 4.9 again in Feature Engineering, or if you don’t follow that, use the material outlined by Towards Science.

Make sure you are reasonably familiar with K-Means, as we are going to implement it in a graph clustering technique known as Spectral Clustering.

# Spectral clustering

## 4.9 Spectral clustering

### What is Spectral clustering?

Spectral clustering is a graph clustering algorithm. It treats each data point as a graph node. Doing this turns the clustering problem into a graph partitioning problem. The concept behind this technique is to transfer the data into a graph and then use a clustering algorithm like K-Means to cluster the graph. The key to it is the use of eigenvalues and eigenvectors and it only requires a few more steps from that shown in Step 4.7. Let’s outline the steps needed to complete a spectral clustering analysis.

### Steps to spectral clustering

#### 1. Building the similarity graph

First, we build a similarity graph in the form of an adjacency matrix. A similarity graph is an unweighted or weighted undirected graph with adjacency matrix, A. The adjacency matrix of an unweighted graph ***G = V, E*** is denoted:

***A = Aij***

where ***Aij = 1*** ***i, j are elements of E,*** otherwise ***0***.

Edges may also have weights ***wij***.

A similarity graph can be built in one of the following ways:

* **ε-Neighbourhood graph:** All points are connected if they lie within the ε-radius. This is an unweighted graph because all of the points lie within a similar scale.
* **K-nearest neighbours:** Here, we use K-nearest neighbors to connect vertex ***vi*** with vertex ***vj*** if ***vi*** is among the k-nearest neighbors of ***vi***.

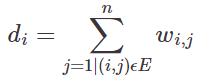
Fully-connected graph: To construct this graph, we simply connect all points with each other, and we weight all edges by similarity ***sij***. Since this approach is used to model the local neighbourhood relationships thus typically the Gaussian similarity metric is used to calculate the distance.

#### 2. Projecting the data onto a low-dimensional space

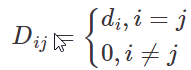
In building the similarity graph, sometimes data points in the same cluster may also be far away, or even farther away, than points in different clusters. So, we need to transform the space so that when the two points are close, they are always in the same cluster, and when they are far apart, they are in different clusters.

This is done by converting the similarity graph to a graph Laplacian matrix, which is another graph representation. To compute it though first, the degree of a node needs to be defined.

The degree of the ***i***th node is given by:



The degree matrix is defined as follows:



So, the Graph Laplacian Matrix is defined as: ***L = D - A***

We calculate the graph Laplacian Matrix to find eigenvalues and eigenvectors for it, in order to reduce the dimensions. The eigenvalues and their eigenvectors are taken and stacked into a matrix such that the eigenvectors are the columns.

#### 3. Clustering the data

This process mainly involves clustering the data by using any traditional clustering technique, e.g. K-Means Clustering.

First, each node is assigned a row of the Graph Laplacian Matrix. Then this data is clustered using any traditional technique. To transform the clustering result, the node identifier is retained.

**Properties:**

**Assumption-less:** This clustering technique, unlike other traditional techniques do not assume the data to follow some property. So, this technique should be used to answer a more-**generic** class of clustering problems.

**Ease of implementation and speed:** This algorithm is easier to implement than other clustering algorithms and is also very fast as it mainly consists of mathematical computations.

**Not-scalable:** Since it involves the building of matrices and computation of eigenvalues and eigenvectors, it **is time-consuming for dense datasets**.

## 4.10 K-medoids clustering

In the previous step, we outlined how the **k-means** algorithm is the most commonly used clustering algorithm. We also outlined how it **has a number of drawbacks related to outliers and the size of the dataset** for example.

K-medoids is a clustering algorithm that uses partitioning to create clusters. Unlike the k-means algorithm (which attempts to minimize the total squared error), the k-medoids choose datapoints as centres. K-medoids is also a partitioning technique of clustering that clusters the data set of ***n*** objects into ***k*** clusters with ***k*** known a priori. It’s **more robust to outliers** than the k-means algorithm an uses the medoid as opposed to the average of a cluster to determine centrality.

* K-means 🡺 Attempts to minimize the total squared error K-medoids using cluster average.

Prone to outliers.

* K-medoids 🡺 uses partitioning to create clusters, choosing datapoints as centres.

More robust to outliers

### Partitioning Around Medoids (PAM)

The most common realisation of k-medoid clustering is the **Partitioning Around Medoids (PAM)** algorithm and is as follows:

1. Initialize: Randomly select ***k*** of the ***n*** data points as the medoids.
2. Assignment step: Associate each data point to the closest medoid.
3. Update step: For each medoid ***m*** and each data point ***o*** associated to ***m*** swap ***m*** and ***o*** and compute the total cost of the configuration (that is, the average dissimilarity of ***o*** to all the data points associated to ***m***). Select the medoid ***o*** with the lowest cost of the configuration.
4. Repeat alternating steps 2 and 3 until there is no change in the assignments.

PAM works well for small datasets and deals with outliers well but has many of the other problems that K-means has. There are other alternatives such as CLARA (Clustering Large Applications) algorithm which select random observations from the dataset and performs partitioning around medoids (PAM) algorithm on them.

The following Google Colab file implements the k-medoids algorithm:

<https://drive.google.com/open?id=1GCGejokDZoYz5bW87LjMpfNkaYV-lAeo>

## 4.11 Spectral clustering worked example

Following the previous section, we will be going through a worked example of spectral clustering using Python code.

In this example, we will be working with an example credit card database from Kaggle, which can be found here:

<https://www.kaggle.com/arjunbhasin2013/ccdata>

Remember the first step that we have to do is create an affinity/similarity matrix. An affinity matrix is just like an adjacency matrix, except the value for a pair of points expresses how similar those points are to each other.

* If pairs of points are very dissimilar 🡺 then the affinity should be 0.
* If the points are identical 🡺 then the affinity might be 1.

You would not consider this example to be a graph problem. This is why we will use techniques such as Radial Based Function or Nearest Neighbour to create the graph. However, if you had a graph then you can easily supply it directly to the Scikitlearn. Or alternatively, you could build the code by mixing the code from Step 4.4.7 with a clustering algorithm such as K-Means.

We will be using scikit-learn to implement the algorithm, and the library allows you to supply your own affinity/similarity matrix. If you are not using your own similarity metric, you can use the function known as a Radial Based Function (RBF). The RBF can be described as a multivariable Gaussian kernel of the Euclidean distance.

Radial basis functions are means to approximate multivariable (also called multivariate) functions by linear combinations of terms based on a single univariate function (the radial basis function). This is radialised so that in can be used in more than one dimension. They are usually applied to approximate functions or data.

If you do not pick the RBF option, it will use the Nearest Neighbour.

I will use both techniques to try and implement the algorithm.

Before we go through the Kaggle example, we will implement the Sckit Learn approach on the data we used in Step 4.7.

Go to the following Google Colab to complete this step.

<https://drive.google.com/open?id=19hVtSS05YYi-awxunksbPgqGGipX0IRY>

# Quiz

**Question 1 –** In graph theory, the objects can be described as edges. True/False?

* True
* False

*Correct, they are known as nodes.*

**Question 2 –** Graphs not only store data about objects but also about the relationships between them. Is this statement true or false?

* True
* False

*Yes, graphs not only store data about objects but also about the relationships between them.*

**Question 3 –** Node A is said to be adjacent to node B if there exists an edge connecting them. Is this statement true or false?

* True
* False

**Question 4 –** The degree is what?

* The number of nodes in the graph.
* The size of the highest eigenvalue from the Laplacian.
* The number of edges connected to a node.

**Question 5 –** Dijkstra’s algorithm can be used to group nodes together on a graph. Is this statement true or false?

* True
* False

**Question 6 –** Structure Databases are typically easier to query. Is this statement true or false?

* True
* False

**Question 7 –** Neo4j is a graph database system. Is this statement true or false?

* True
* False

**Question 8 –** An adjacency matrix always has values of either 0 or 1. Is this statement true or false?

* True
* False

*This is only the case on a finite simple matrix.*

**Question 9 –** In spectral clustering, if all the eigenvalues are 0 then we have a completely connected graph. Is this statement true or false?

* True
* False

*If the eigenvalue A equals 0 then Ax = 0x = 0. Vectors with eigenvalue 0 make up the nullspace of A.*

**Question 10 –** A Radial Basis Function (RBF) uses a Gaussian kernel on a euclidean metric. Is this statement true or false?

* True
* False