# Tokenization

Tokenization is a common task in Natural Language Processing (NLP). It’s a fundamental step in both traditional NLP methods like Count Vectorizer and Advanced Deep Learning-based architectures like Transformers.

Tokenization is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either words, characters, or subwords. Hence, tokenization can be broadly classified into 3 types – word, character, and subword (n-gram characters) tokenization.

The most common way of forming tokens is based on space. Assuming space as a delimiter, the tokenization of the sentence results in 3 tokens – Never-give-up. As each token is a word, it becomes an example of Word tokenization.

Similarly, tokens can be either characters or subwords. For example, let us consider “smarter”:

Character tokens: s-m-a-r-t-e-r

Subword tokens: smart-er

Word Tokenization is the most used tokenization algorithm. It splits a piece of text into individual words based on a certain delimiter. Depending upon delimiters, different word-level tokens are formed. Pretrained Word Embeddings such as Word2Vec and GloVe comes under word tokenization.

# Lemmatization

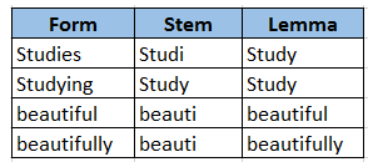
Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meaning to one word.

Text preprocessing includes both Stemming as well as Lemmatization. Many times people find these two terms confusing. Some treat these two as same. Lemmatization is preferred over Stemming because lemmatization does morphological analysis of the words.

Applications of lemmatization are:

* Used in comprehensive retrieval systems like search engines.
* Used in compact indexing

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the “lemma”. If confronted with the token “saw”, stemming might return just s, whereas lemmatization would attempt to return either “see” or “saw” depending on whether the use of the token was as a verb or a noun. The two may also differ in that stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma. Linguistic processing for stemming or lemmatization is often done by an additional plug-in component to the indexing process, and a number of such components exist, both commercial and open-source.



# Stop Words

Stopwords are the English words which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. For example, the words like the, he, have etc. Search engines have been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

Input: “This is a sample sentence, showing off the stop words filtration.”

Output: ['This', 'sample', 'sentence', 'showing', 'stop', 'words', 'filtration']

# Word Vectorization

It is a general process of turning a collection of text documents into numerical feature vectors. There are many methods to convert text data to vectors which the model can understand but by far the most popular method is called TF-IDF. This is an acronym than stands for “Term Frequency — Inverse Document Frequency”.

TF-IDF is frequently used in machine learning algorithms in various capacities, including stop-word removal. These are common words like “a, the, an, it” that occur frequently but hold little informational value. TF-IDF consists of two components, term frequency, and inverse document frequency.

In a simple language, TF-IDF can be defined as follows:

A High weight in TF-IDF is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents.

TF-IDF algorithm is made of 2 algorithms multiplied together.

* Term Frequency
  + Term frequency (TF) is how often a word appears in a document, divided by how many words there are.
  + TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)
* Inverse document frequency
  + Term frequency is how common a word is, inverse document frequency (IDF) is how unique or rare a word is.
  + IDF(t) = log\_e(Total number of documents / Number of documents with term t in it)

# SVM

Reference: https://www.upgrad.com/blog/support-vector-machines/

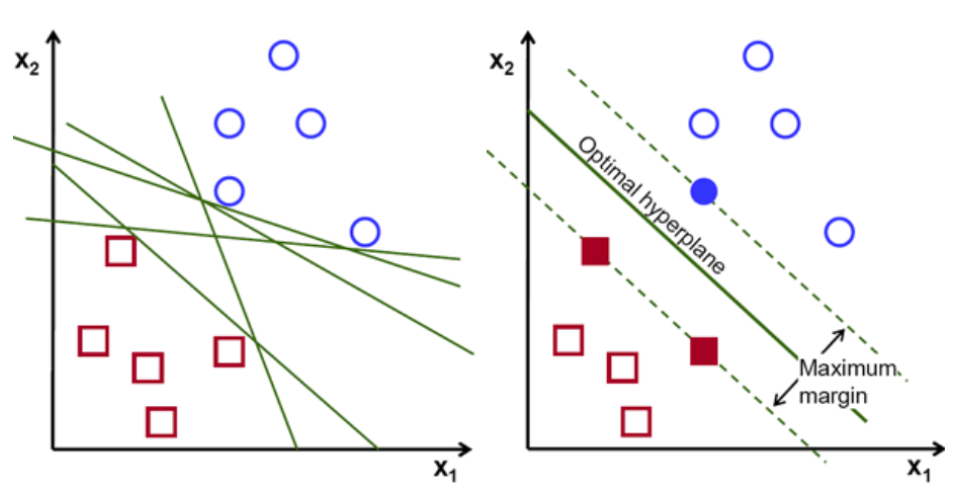
A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be employed for both classification and regression purposes. SVMs are based on the idea of finding a hyperplane that best divides a dataset into two classes.

**Types of SVM**

Linear SVM : Linear SVM is used for data that are linearly separable i.e. for a dataset that can be categorized into two categories by utilizing a single straight line. Such data points are termed as linearly separable data, and the classifier is used described as a Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for data that are non-linearly separable data i.e. a straight line cannot be used to classify the dataset. For this, we use something known as a kernel trick that sets data points in a higher dimension where they can be separated using planes or other mathematical functions. Such data points are termed as non-linear data, and the classifier used is termed as a Non-linear SVM classifier.

Firstly, set of points belonging to the two classes are plotted and visualized as shown below. In a 2-d space by just applying a straight line, we can efficiently divide these two classes. But there can be many lines that can classify these classes. There are a set of lines or hyperplanes(green lines) to choose from. The question will be, out of all these lines which line is suitable for classification?



Basically, select the hyper-plane which separates the two classes better. We do this by maximizing the distance between the closest data point and the hyper-plane. The greater the distance, the better is the hyperplane and better classification results ensue. It can be seen in the figure below that the hyperplane selected has the maximum distance from the nearest point from each of those classes.

A reminder, the two dotted lines that go parallel to the hyperplane crossing the nearest points of each of the classes are referred to as the support vectors of the hyperplane. Now, the distance of separation between the supporting vectors and the hyperplane is called a margin. And the purpose of the SVM algorithm is to maximize this margin. The optimal hyperplane is the hyperplane with maximum margin.

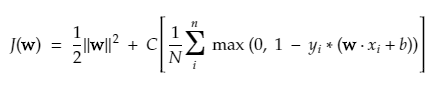
Take for example classifying cells as good and bad. the cell xᵢ is defined as an n-dimensional feature vector that can be plotted on n-dimensional space. Each of these feature vectors are labeled with a class yᵢ. The class yᵢ can either be a +ve or -ve (eg. good=1, not good =-1). The equation of the hyperplane is y=w.x + b = 0. Where W and b are line parameters. The earlier equation returns a value ≥ 1 for examples for +ve class and ≤-1 for -ve class examples.

The hyperplane is defined by finding the optimal values w or weights and b or intercept which. And these optimal values are found by minimizing the cost function. Once the algorithm collects these optimal values, the SVM model or the line function f(x) efficiently classifies the two classes.

In a nutshell, the optimal hyperplane has equation w.x+b = 0. The left support vector has equation w.x+b=-1 and the right support vector has w.x+b=1.

Thus the distance d between two parallel liens Ay = Bx + c1 and Ay = Bx + c2 is given by d = |C1–C2|/√A^2 + B^2. With this formula in place, we have the distance between the two support vectors as 2/||w||.

The cost function for SVM looks the like the equation below:



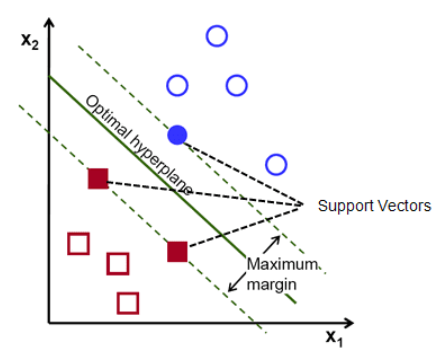
In the cost function equation above, the λ parameter denotes that a larger λ provides a broader margin, and a smaller λ would yield a smaller margin. Furthermore, the gradient of the cost function is calculated and the weights are updated in the direction that lowers the lost function.

**Algorithm for Non-linear SVM**

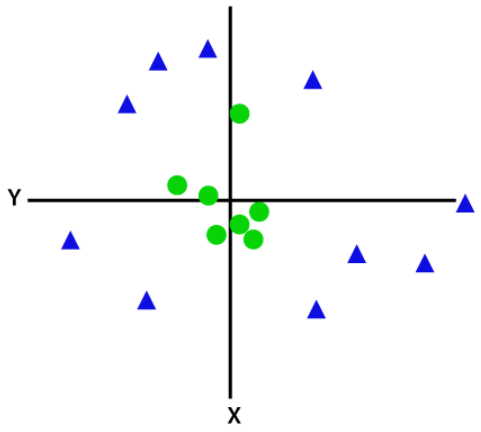
In the SVM classifier, it is straight forward to have a linear hyper-plane between these two classes. But, an interesting question which arises is, what if the data is not linearly separable, what should be done? For this, the SVM algorithm has a method called the kernel trick.

The SVM kernel function takes in low dimensional input space and converts it to a higher-dimensional space. In simple words, it converts the not separable problem to a separable problem. It performs complex data transformations based on the labels or outputs that define them

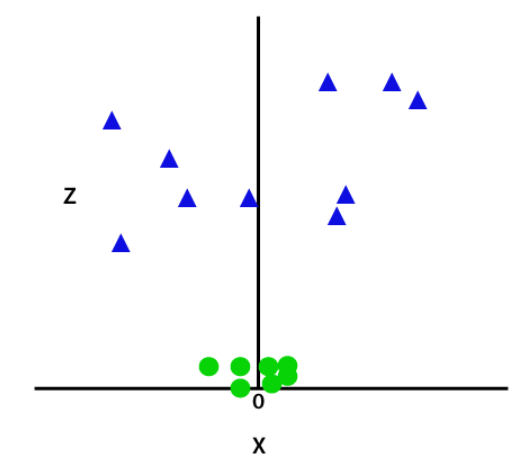
Look at the diagram below to better understand data transformation. The set of data points on the left are clearly not linearly separable. But when we apply a function Φ to the set of data points, we get transformed data points in a higher dimension that is separable via a plane.



To separate non linearly separable data points, we have to add an extra dimension. For linear data, two dimensions have been used, that is, x and y. For these data points, we add a third dimension, say z. For the example below let z=x² +y².



This z function or the added dimensionality transforms the the sample space and the above image will become as the following:

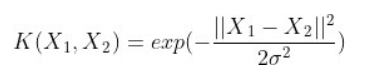


On close analysis, it is evident that the above data points can be separated using a straight-line function that is either parallel to the x axis or is inclined at an angle. Different types of kernel functions are present — linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.

# RBF Kernel

Reference: <https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a>

Radial Basis Function (RBF) kernels are the most generalized form of kernelization and is one of the most widely used kernels due to its similarity to the Gaussian distribution. The RBF kernel function for two points X₁ and X₂ computes the similarity or how close they are to each other. This kernel can be mathematically represented as follows:

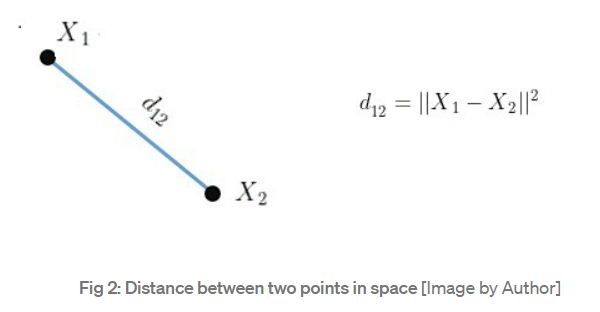


where,

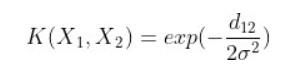
1. ‘σ’ is the variance and our hyperparameter

2. ||X₁ - X₂|| is the Euclidean (L₂-norm) Distance between two points X₁ and X₂

Let d₁₂ be the distance between the two points X₁ and X₂, we can now represent d₁₂ as follows:



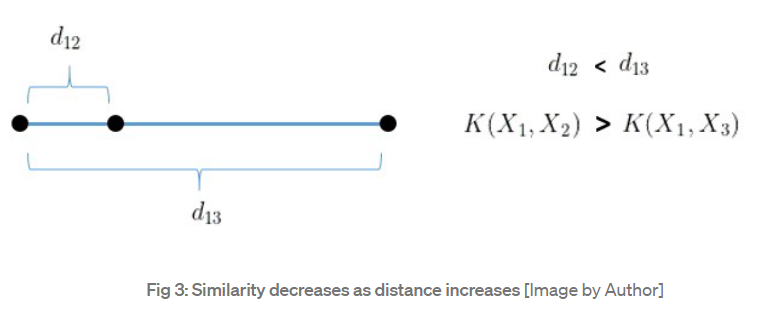
The kernel equation can be re-written as follows:



The maximum value that the RBF kernel can be is 1 and occurs when d₁₂ is 0 which is when the points are the same, i.e. X₁ = X₂.

* When the points are the same, there is no distance between them and therefore they are extremely similar
* When the points are separated by a large distance, then the kernel value is less than 1 and close to 0 which would mean that the points are dissimilar

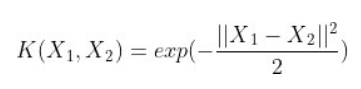
Distance can be thought of as an equivalent to dissimilarity because we can notice that when distance between the points increases, they are less similar.



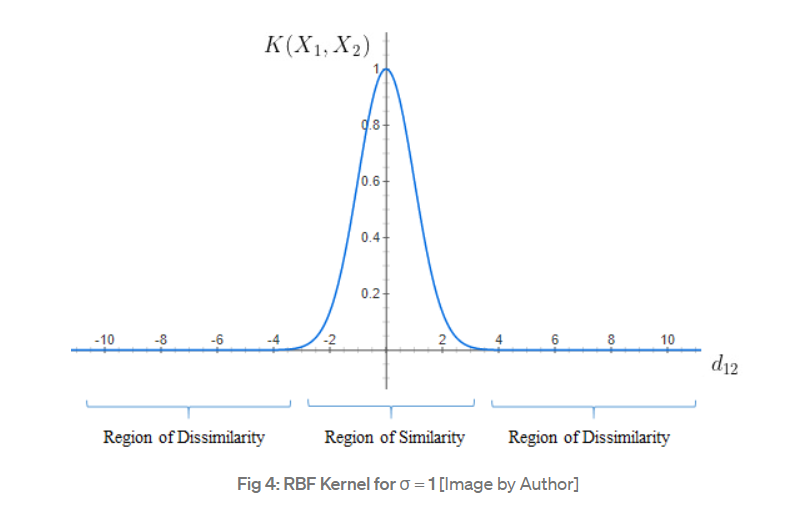
It is important to find the right value of ‘σ’ to decide which points should be considered similar and this can be demonstrated on a case by case basis.

**a] σ = 1**

When σ = 1, σ² = 1 and the RBF kernel’s mathematical equation will be as follows:



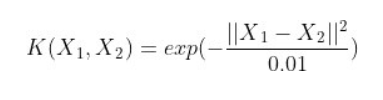
The curve for this equation is given below and we can notice that as the distance increases, the RBF Kernel decreases exponentially and is 0 for distances greater than 4.



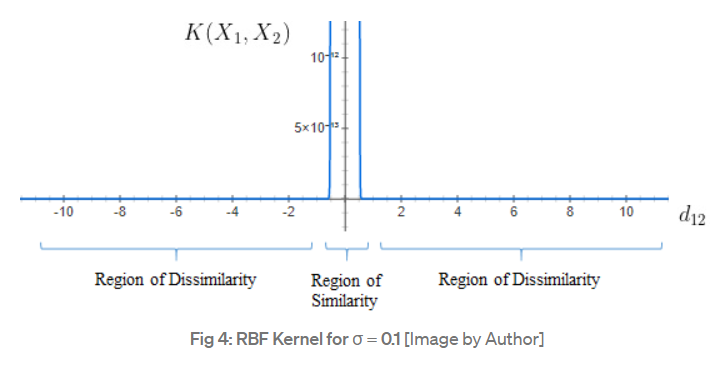
* We can notice that when d₁₂ = 0, the similarity is 1 and as d₁₂ increases beyond 4 units, the similarity is 0
* From the graph, we see that if the distance is below 4, the points can be considered similar and if the distance is greater than 4 then the points are dissimilar

**b] σ = 0.1**

When σ = 0.1, σ² = 0.01 and the RBF kernel’s mathematical equation will be as follows:



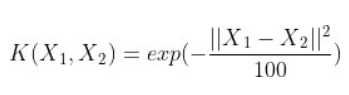
The width of the Region of Similarity is minimal for σ = 0.1 and hence, only if points are extremely close they are considered similar.



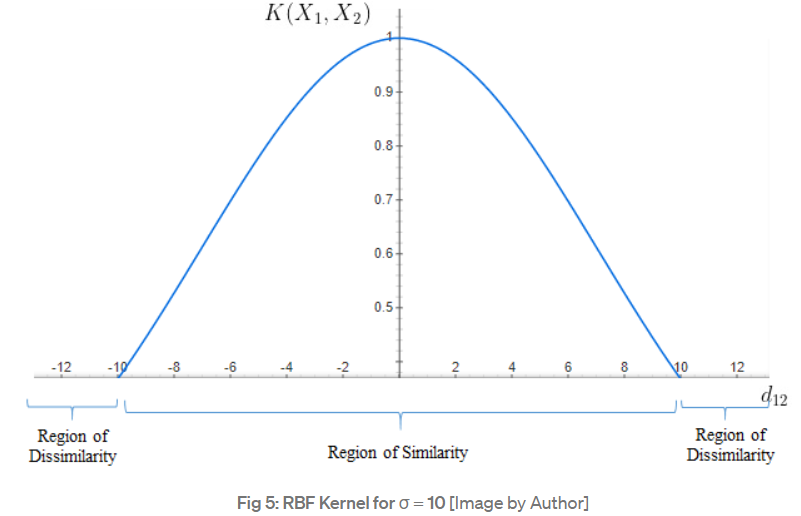
* We see that the curve is extremely peaked and is 0 for distances greater than 0.2
* The points are considered similar only if the distance is less than or equal to 0.2

**b] σ = 10**

When σ = 10, σ² = 100 and the RBF kernel’s mathematical equation will be as follows:



The width of the Region of Similarity is large for σ = 100 because of which the points that are farther away can be considered to be similar.



* The width of the curve is large
* The points are considered similar for distances up to 10 units and beyond 10 units they are dissimilar