# TOXIC COMMENT CLASSIFICATION

# ABSTRACT

Text classification has become one of the most useful applications this process includes techniques like Tokenizing, Stemming, and multi classification. This project uses NLP technique and SVM algorithm, that are used to classify online comments based on their level of toxicity.

## CHAPTER – 1

## INTRODUCTION

Social media is a place where a lot of discussions happen, being anonymous while doing so has given the freedom to many people to express their opinions freely. But people who disagree with a point of view extremely can misuse this freedom sometimes. Sharing things that you care about will become a difficult task with this constant threat of harassment or toxic comments online.

This will eventually lead to people not sharing their ideas online and stop asking for other people’s opinion on them. Unfortunately, the social media platforms face these issues all the time and find it difficult to identify and stop these toxic remarks before it leads to the abrupt end of conversations.

In this project using Natural Language Processing with regression technique to solve this problem of identifying the toxicity of online comments. Word embeddings will be used in conjunction with recurrent with Support Vector Machine (SVM).

Toxic comments are defined as comments that are rude, disrespectful, or that tend to force users to leave the discussion. If these toxic comment can be auto matically identified, we could have safer discussions on various social networks, news portals, or online forums. Manual moderation of comments is costly, ineffective, and sometimes infeasible.

Automatic or semi-automatic detection of toxic comment is done by using different machine learning methods, mostly different deep neural networks architectures. Recently, there is a significant number of research papers on the toxic comment classification problem, but, to date, there has not been a systematic literature review of this research theme, making it difficult to assess the maturity, trends and research gaps.

## CHAPTER - 2

## PROBLEM SPECIFICATION

**Existing System:**

Building a multi-headed model that's capable of detecting different types of toxicity like threats, obscenity, insult and identity-based hate by using logistic regression.

**Proposed System:**

In this project a Support vector machine model to classify the comments and calculate the model’s accuracy. The comments are first passed to a tokenizer or vectorizer to create a dictionary of words, then an embedding matrix is created after which it is passed to a model to classify.

**Objective:**

* To develop a **Multi-Label Toxic comments Classification**
* To implement Machine Learning Algorithm Support Vector Regression by kernel pipeline
* Calculating accuracy of each type of toxic.

## CHAPTER - 3

## SOFTWARE REQUIREMENTS SPECIFICATIONS

**Functional Requirements:**

The functional requirements for a system describe what the system should do. Those requirements depend on the type of software being developed, the expected users of the software. These are statement of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situation.

* NLP is used For Comment words separation, in these comments npl uses these techniques tokenization, stemming and vectorization.
* This model is used to Classify the all toxic comments category for a given dataset.
* When the system has any input value being missed, it could not provide accurate results.
* From the input dataset, there are 8 attributes that defines in that 6 are the categories of the comments and reaming are the comment and Id, of which having of 159571 tuples, that define the classification of the toxic comments are considered.

**Non-Functional Requirements:**

Nonfunctional requirements are requirements that are not directly concerned with the specified function delivered by the system. They may relate to emergent system properties such as reliability, response time and store occupancy.

Some of the non-functional requirements related with this system are hereby below:

* **Scalability**: The model is designed to pre-process the data and deliver correct results varying with the scalable data
* **Accuracy**: The model uses SVM machine learning algorithm for providing accurate results based on input data
* **Serviceability**: This project is used know the type of comment and this service is we can provide to all social media platform.
* **Reliability**: Accuracy Scores that are highly reliable are precise, reproducible, and consistent from testing set.

**Software Requirements:**

**Operating System:** Windows 10

**IDE:** Jupyter Notebook, Anaconda3

**Language:** Python

**Framework:** Flask

**Windows OS:**

Windows OS, a computer operating system (OS) developed by Microsoft Corporation to run personal computers. Featuring the first graphical user interface for IBM-compatible PCs, the Windows OS soon dominated the PC market. Approximately 90 percent of PCs run some version of the Windows OS.

**Hardware Requirements:**

**Processor:** i3 Processor or above

**RAM:** 4 GB or above

**Hard Disk:** 512 GB or above

**Feasibility Study:**

Before starting the project, feasibility study is carried out to measure the viable of the system. Feasibility study is necessary to determine if creating a new or improved system is friendly with the cost, benefits, operation, technology and time. Following feasibility study is given as below:

1. **Technical Feasibility-**

Technical feasibility is one of the first studies that must be conducted after the project has been identified. Technical feasibility study includes the hardware and software devices.

The required technology (Python language and Jupyter Notebook) exists.

1. **Operational Feasibility-**

Operational Feasibility is a measure of how well a proposed system solves the problem and takes advantage of the opportunities identified during scope definition. The following points were considered for the project’s technical feasibility:

1. The system will take input CSV data and SVR model to execute.
2. The model the returns the result as toxic classifications of comments
3. **Economic Feasibility-**

The purpose of economic feasibility is to determine the positive economic benefits that include quantification and identification. The system is economically feasible due to availability of all requirements such as collection of data from internet. The system is time effective as it takes less time to classify the comments. The result generated is accurate.

1. **Schedule Feasibility-**

Schedule feasibility is a measure of how reasonable the project timetable is. The system is found schedule feasible because the system is designed in such a way that it will finish with in prescribed time.

1. **Behavioral Feasibility-**

The system working is quite easy to use and learn due to its simple but attractive interface. User requires no special training for operating the system.

## CHAPTER - 4

## SYSTEM DESIGN

## 4.1 System Architecture

System architecture is a conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structure and behavior of the system. A representation of a system, including a mapping of functionally onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and human interaction with these components.

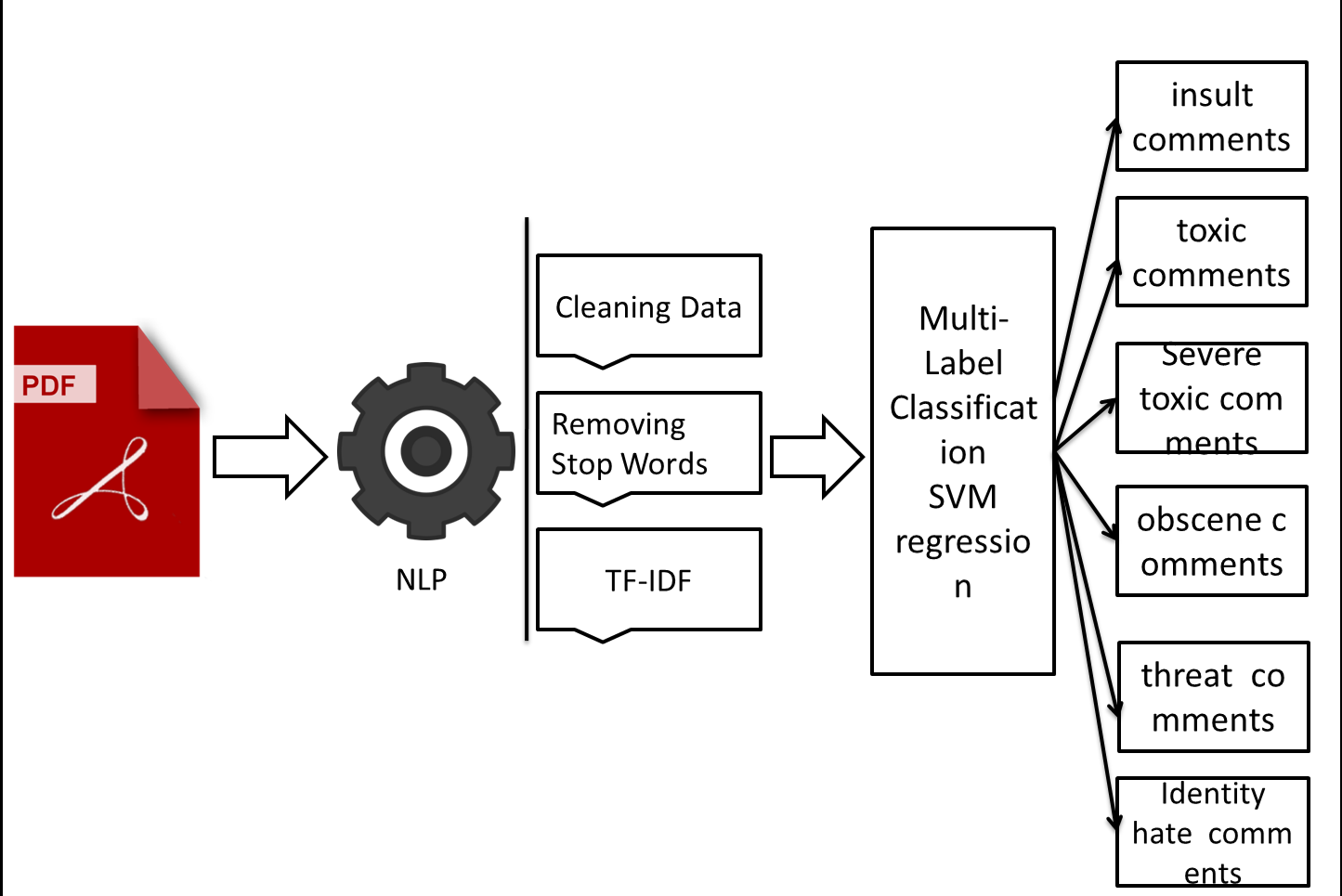


Fig 4.1 System architecture

## CHAPTER - 5

## DATASET

## Introduction:

A dataset is a collection of data. Most commonly a dataset corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable, and each row corresponds to a given member of the dataset in question. The data set lists values for each of the variables such as height or weight of an object for each member in the dataset. A data set is organized into some type of data structure. In a database, for example, a data set might contain a collection of business data (names, salaries, contact information, sales figures, and so forth). The database itself can be considered a data set, as can bodies of data within it related to a particular type of information, such as sales data for a particular corporate department.

## Dataset Description:

You are provided with a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The types of toxicity are:

* toxic
* severe\_toxic
* obscene
* threat
* insult
* identity\_hate

In this dataset consists of 8 columns and 159571 records.

**File descriptions**

* **train.csv** - the training set, contains comments with their binary labels
* **test.csv** - the test set, you must predict the toxicity probabilities for these comments. To deter hand labeling, the test set contains some comments which are not included in scoring.
* **sample\_submission.csv** - a sample submission file in the correct format
* **test\_labels.csv** - labels for the test data; value of -1 indicates it was not used for scoring; (Note: file added after competition close!)

## Data Pre -Processing:

Data Pre-Processing, is a technique which is used to transform the raw data in a useful and efficient format. In this project data preprocessing is applied for removing NULL values. Data preprocessing includes different tasks as data cleaning, feature selection and data transformation.

**Data Visualization:**

Data visualization is another form of visual art that grabs people’s interest and keeps their eyes on the message. When a person sees a chart then he/she can quickly see trends and outliers. If a person can see something, he/she internalize it quickly. It’s storytelling with a purpose. Data can be visualized in a number of forms such as: Histograms, Density Plots, Correlation Matrix Plots, Pie Chart etc.,.

All these are showed in **Chapter 10**

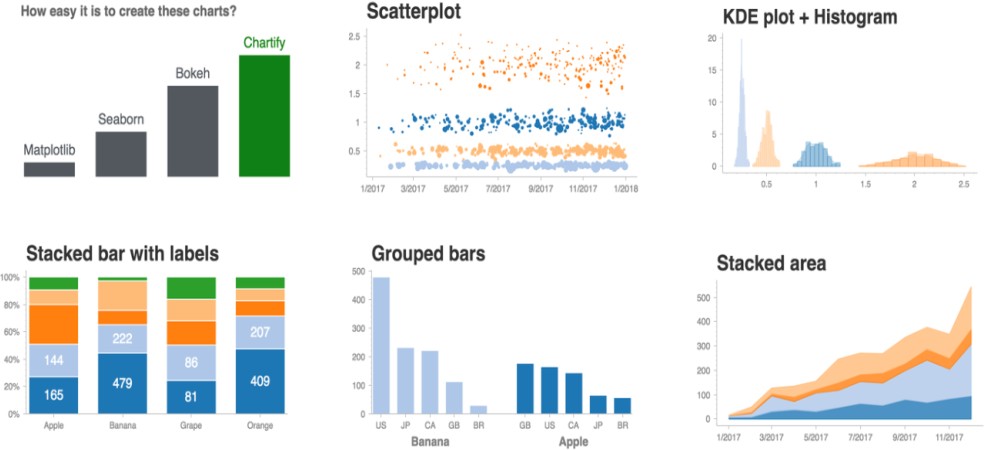


Fig 5.1 Data Visualization

## Dataset Splitting:

The fundamental goal of an ML model is to make accurate predictions on future data instances beyond those used to train models. Before using an ML model to make predictions, one need to evaluate the predictive performance of the model. To estimate the quality of an ML model predictions with data it has not seen, one can reserve, or split, a portion of the data for which he/she already know the answer as a proxy for future data and evaluate how well the ML model predicts the correct answers for that data. The dataset can be split into a portion for the training and a portion for the evaluation

### Train Set:

In this project taking 75% of data taking as training data.

**Test Set:**

In this project taking 25% of data taking as testing data.

## CHAPTER – 6

## MODELING

## Introduction to Machine Learning:

Machine learning is one of the fields in artificial intelligence. It is the scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task. It uses the statically technique to give the computer systems the ability to learn from the given data without being explicitly programmed. Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly. Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. Combining machine learning with AI and cognitive technologies can make it even more effective in processing large volumes of information. To predict the diabetes classification the most popular technique is Classification which is supervised learning. The algorithms implemented in this project are: Random Forest & Support Vector Machines Algorithms.

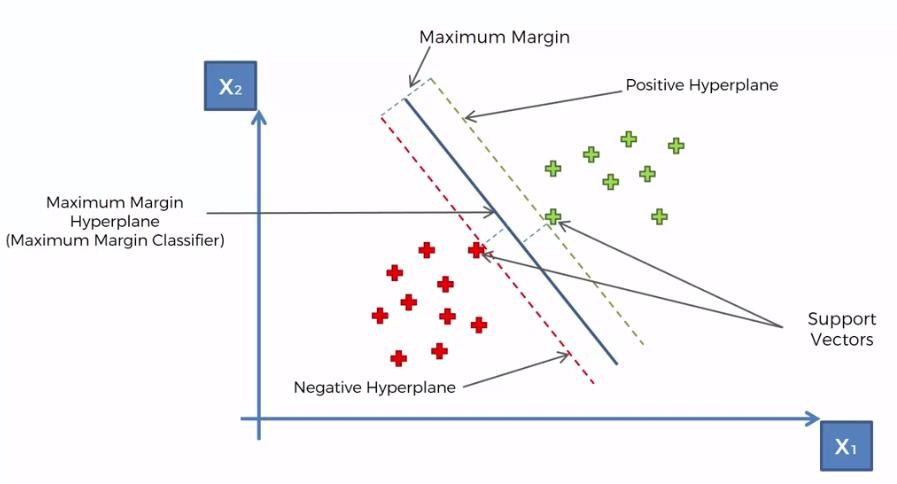
## SUPERVISED LEARNING

Supervised learning is a learning model built to make prediction, given an unforeseen input instance. A supervised learning algorithm takes a known set of input dataset and its known responses to the data (output) to learn the regression/classification model. A learning algorithm then trains a model to generate a prediction for the response to new data or the test dataset. Supervised learning uses classification algorithms and regression techniques to develop predictive models. The algorithms include linear regression, logistic regression, and neural networks as well, apart from decision tree, Support Vector Machine (SVM), random forest, naive Bayes, and k-nearest neighbour. Classification task predicts discrete responses. It is recommended if the data can be categorized, tagged, or separated into specific groups or classes. Classification models classify input data into categories. Regression techniques predict continuous responses. A linear regression attempts to model the relationship between two variables by fitting linear equation to observed data.

In Supervised learning, one trains the machine using data which is well "labeled." It means some data is already tagged with the correct answer. It can be compared to learning which takes place in the presence of a supervisor or a teacher. A supervised learning algorithm learns from labelled training data, helps you to predict outcomes for unforeseen data. Successfully building, scaling, and deploying accurate supervised machine learning models takes time and technical expertise from a team of highly skilled data scientists. Moreover, Data scientist must rebuild models to make sure the insights given remains true until its data changes.

## Support Vector Machine:

The maximal margin classifier is a very natural way to perform classification, is a separating hyperplane exists. However, the existence of such a hyperplane may not be guaranteed, or even if it exists, the data is noisy so that maximal margin classifier provides a poor solution. In such cases, the concept can be extended where a hyperplane exists which almost separates the classes, using what is known as a soft margin. The generalization of the maximal margin classifier to the non-separable case is known as the support vector classifier, where a small proportion of the training sample is allowed to cross the margins or even the separating hyperplane. Rather than looking for the largest possible margin so that every observation is on the correct side of the margin, thereby making the margins very narrow or non-existent, some observations are allowed to be on the incorrect side of the margins. The margin is soft as a small number of observations violate the margin. The softness is controlled by slack variables which control the position of the observations relative to the margins and separating hyperplane. The support vector classifier maximizes a soft margin.

Fig 6.1 Support Vector Machine

**Algorithm:**

Input: Set of data points

Output: Hyper Plane classifying the points Steps:

Step 1: *A hyperplane is drawn that best classifies the data into classes and assume that all the data is perfectly classified.*

Step 2: *Calculate the perpendicular distance from the closest data points to the hyperplane. This is known as margin. An optimal hyperplane should be such that the margin is maximum.*

Step 3: *For a new data point, to be classified into the class to which it belongs, consider the following equation*

𝐹(𝑥) = 𝐵(0) + 𝑠𝑢𝑚( 𝐵(𝑖) ∗ (𝑥, 𝑥(𝑖)) ) , 𝑖 = 1, 2, 3, ….

*This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients* (0) *and* 𝐵(𝑖) *(for each input) must be estimated from the training data by the learning algorithm.*

Step 3.a: *If* (𝑥) > 0 *(is greater than zero), then the new data point(input) is above the plane.*

Step 3.b: *If* (𝑥) < 0*(is less than zero), then the new data point(input) is below the plane*.

Step 3.c: *A value close to the line returns a value close to zero and the point may be difficult to classify.*

Step 3.d: *If the magnitude of the value is large, the model may have more confidence in the prediction.*

Step 4: *The best or optimal plane is the one that has the maximum margin.*

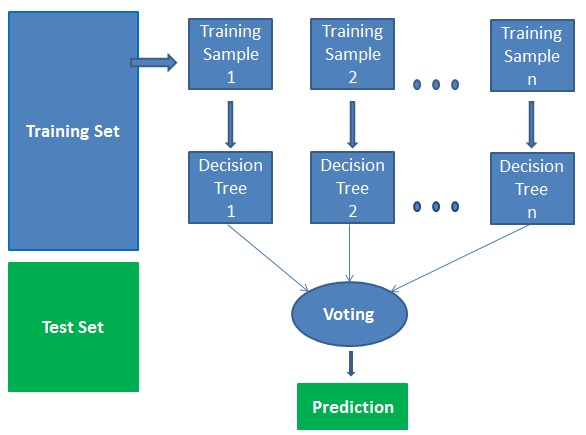
## RANDOM FOREST ALOGRITHM

Random forest is a supervised learning algorithm which is used for both classification as well as regression. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result. Random forest is nothing but a collection of multiple decision tree models. Random forest is a supervised Machine Learning algorithm. This algorithm creates a set of decision trees from a few randomly selected subsets of the training set and picks predictions from each tree. Then by means of voting, the random forest algorithm selects the best solution. The [random forest](https://en.wikipedia.org/wiki/Random_forest) algorithm combines multiple algorithm of the same type i.e. multiple decision trees, resulting in a forest of trees, hence the name "Random Forest". The random forest algorithm can be used for both regression and classification tasks.

Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.Random forests has a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases.

It works in four steps:

1. Select random samples from a given dataset.
2. Construct a decision tree for each sample and get a prediction result from each decision tree.
3. Perform a vote for each predicted result.
4. Select the prediction result with the most votes as the final prediction.



**Fig 7.1**. Random Forest

**Advantages:**

* Random forests are considered as a highly accurate and robust method because of the number of decision trees participating in the process.
* It does not suffer from the over fitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases.
* The algorithm can be used in both classification and regression problems.
* Random forests can also handle missing values. There are two ways to handle these: using median values to replace continuous variables, and computing the proximity-weighted average of missing values.
* You can get the relative feature importance, which helps in selecting the most contributing features for the classifier.

**Disadvantages:**

* Random forests is slow in generating predictions because it has multiple decision trees. Whenever it makes a prediction, all the trees in the forest have to make a prediction for the same given input and then perform voting on it. This whole process is time-consuming.
* The model is difficult to interpret compared to a decision tree, where you can easily make a decision by following the path in the tree.

**Finding important features**

Random forests also offers a good feature selection indicator. Scikit-learn provides an extra variable with the model, which shows the relative importance or contribution of each feature in the prediction. It automatically computes the relevance score of each feature in the training phase. Then it scales the relevance down so that the sum of all scores is 1. This score will help you choose the most important features and drop the least important ones for model building.

Random forest uses gini importance or mean decrease in impurity (MDI) to calculate the importance of each feature. Gini importance is also known as the total decrease in node impurity. This is how much the model fit or accuracy decreases when you drop a variable. The larger the decrease, the more significant the variable is. Here, the mean decrease is a significant parameter for variable selection. The Gini index can describe the overall explanatory power of the variables.

**Random Forests vs Decision Trees**

* Random forests is a set of multiple decision trees.
* Deep decision trees may suffer from overfitting, but random forests prevents over fitting by creating trees on random subsets.
* Decision trees are computationally faster.
* Random forest is difficult to interpret, while a decision tree is easily interpretable and can be converted to rules.

# CHAPTER - 7

# METHODOLOGY

**7.1 Information extraction**

The first phase of our proposed system involves information extraction using Natural Language Processing. The information in the resumes is not present in a structured format. There are noises, inconsistencies and irrelevant bits of data which is of no use to the recruiters. The objective is to derive relevant keywords from the unstructured textual data in the resume without any need of human crawling efforts. Using techniques like Tokenization, Stemming, POS Tagging, Named Entity Recognition, etc., our system obtains important job-related content (skills, experience, education, etc.) from the uploaded candidate resumes. The result is a summarised version of each resume in a JSON format which can be easily used for further processing tasks in the next phase of this resume screening system.

**7.1.1 Tokenization**

After converting the various resume formats (.docx, .pdf, .jpg, .rtf, etc.) into text, we begin the tokenization process to identify terms or words that form up a character sequence. This is important as through these words, we will be able to derive meaning from the original text sequence. Tokenization involves dividing big chunks of text into smaller parts called tokens. This is done by removing or isolating characters like whitespaces and punctuation characters. Tokens are sentences initially (when tokenized out of paragraphs) and then are further split into individual words. By performing Tokenization, we can derive information like the number of words in a text, frequency of a particular word in the text and much more. The tokenization can be performed in multiple ways such as using Natural Language Toolkit [NLTK], the spaCy library, etc. Tokenization is a mandatory step for further text processing such as removal of stop words, stemming and lemmatization

**7.1.2 Stemming and lemmatization**

It is frequently seen that a single word of the English language is used in various different forms in different sentences according to its grammatical rules. For example -implement, implemented and implementing are just different tenses of the same verb. This situation results in the need to reduce all the altered or derived forms of a word to their central stem or base so that these derivationally related words with similar meanings are not considered to be different from each other. Both Stemming and lemmatization have the same objective but differ in their approach.

“Stemming is the mechanism of reducing inflected or derived words to their word root, or stem. It is a crude heuristic process that involves chopping off the ends of words to achieve this objective, and often includes the removal of derivational affixes” (Jivani, A.G., 2011). These are rulebased algorithms in which a particular word is tested on a range of conditions and then based on a list of known suffixes, decides how to cut it down. It is noteworthy that the root derived after stemming may not be identical to the morphological root of the word. Due to the heuristic-based approach of stemming, it suffers from issues such as under-stemming and over-stemming. Some common stemming algorithms used are PorterStemmer, Snowball stemmer, and Lancaster stemmer. On the other hand, lemmatization is the process of utilising a language dictionary to perform an accurate reduction to root words. Unlike Stemming which simply cuts off tokens by simple pattern matching, lemmatization is a more careful approach that uses language vocabulary and morphological analysis of words to give linguistically correct lemmas. This means lemmatization utilises the knowledge of context and therefore can differentiate between words that have different meanings based on parts of speech. For the English language, our system uses the WordNet Lemmatizer (based on WordNew Database) provided by the NLTK python package.

**7.1.3 Parts of speech (POS) tagging**

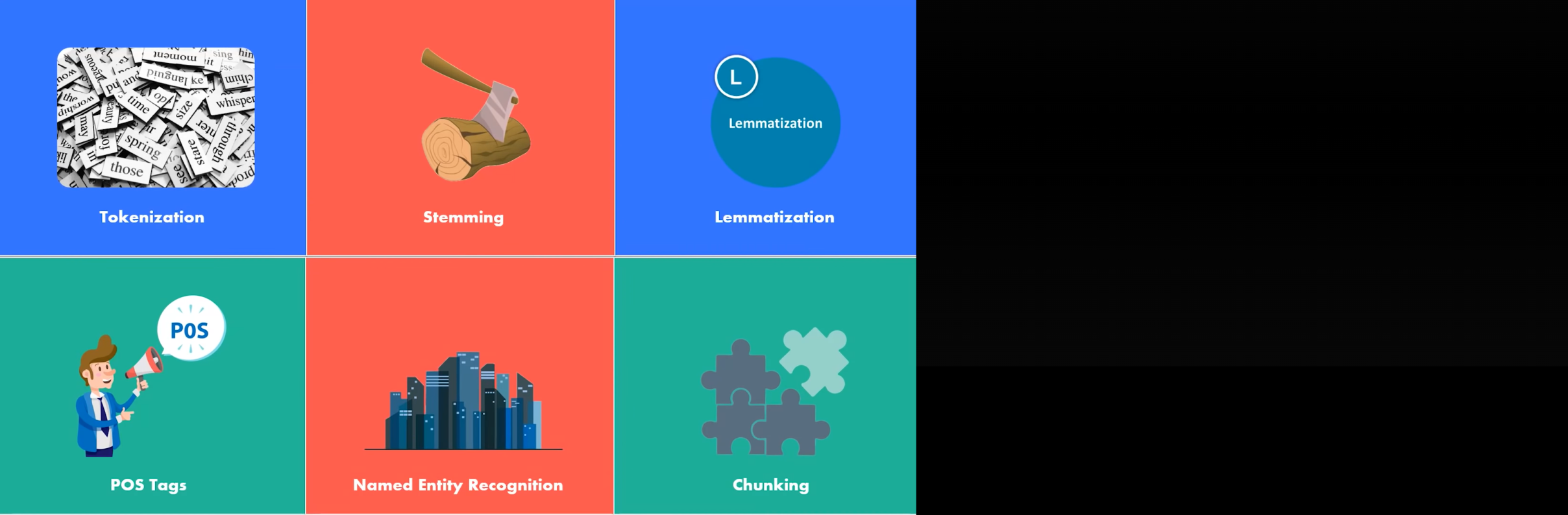
It is a process of assigning grammatical information to a word based on its context and its relationship with other words in the sentence (Gelbukh, 2014). The part-of-speech tag specifies whether the word is a noun, pronoun, verb, adjective, etc. according to its usage in the sentence. It is important to assign these tags so as to understand the correct meaning of a sentence and for building knowledge graphs for named entity recognition. This process is not as simple as mapping a word to their corresponding part of speech tags. This is so as a particular word may have a different part of speech based on different contexts in which it is used. For example: In the sentence “I am building a software”, building is a Verb, but in the sentence “I work in the tallest building of that street”, building is a Noun. Also called grammatical tagging or word-category disambiguation, it is a supervised learning solution that analyses the features such as the preceding word, following word, first letter capitalized or not, etc. to label the words after tokenization. Rule-Based POS tagging, Stochastic POS tagging, and Transformation based tagging are mostly used (Hasan, 2006).

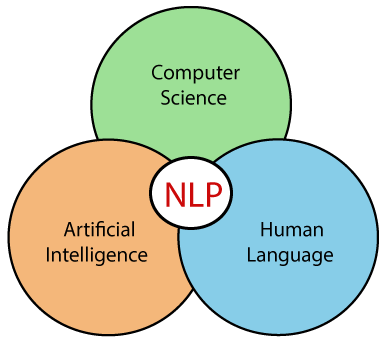
**7.1.4 Chunking**

Chunking is a process that aims to add more structure to sentences by grouping short phrases with parts of speech tags. Because parts of speech tags alone cannot give information about the structure of the sentence or the actual meaning of the text, chunking combines parts of speech tags with regular expressions to give a result as a set of chunk tags like Noun Phrase (NP), Verb Phrase (VP), etc. Also called Shallow Parsing, it involves the construction of a parse tree that can have a maximum one level of information from roots to leaves. This ensures there is more information than just part of speech of the word without needing to create a full parse tree. Chunking segments and labels multi-token sequences (Bird, Klein and Loper, 2009), mostly making groups of “noun phrases” that are used for finding named entities.

**7.1.5 Named entity recognition**

Named Entity Recognition is an information extraction technique which extracts relevant information by classifying chunks of unorganized text into predefined categories like names of persons, companies, contact info, educational credentials, and skills. After classifying the unstructured resume data into such different sets of categories, our aim is to use a similarity model to determine the similarity between the categorized resume data and the requirements provided by the recruiters. There are many approaches to implement the Named Entity Recognition (Mansouri, A., Affendey, L.S. and Mamat, A., 2008) in order to derive relevant categories from unstructured data. These include the Rule-Based approach in which we define our own algorithms according to the required domain. We can also use regular expressions, which finds patterns in a string to detect the named entities. Another approach is using Bidirectional-LSTM with the Conditional Random Field algorithm for named entity recognition as a sequence labelling problem (Huang, Z., Xu, W. and Yu, K., 2015). We have used the spaCy module which consists of various pre-trained models that can recognize a number of default entities from the content of the documents. These models use language information to detect these entities. We also trained the model on a large annotated set of resume samples for better accuracy in the entity recognition. We could detect entities like name, phone number, email, educational institute, organisation etc. from the resumes.





**7.2 Content based candidate recommendation**

The second phase of our proposed system aims to build a content-based recommendation system (Guo, X., Jerbi, H. and O'Mahony, M.P., 2014) that utilises the extracted entities from phase 1 to recommend the most appropriate resumes for the given job description. The system employs concepts like Vectorisation (Salton, G., Wong, A. and Yang, C.S., 1975), importance or weight assigning techniques like TF-IDF (Jabri, Siham, et al., 2018) and similarity measures like cosine distance (Huang and Anna, 2008) for calculating the similarity among the contents of the documents.

**7.2.1 Content based candidate recommendation**

The second phase of our proposed system aims to build a content-based recommendation system (Guo, X., Jerbi, H. and O'Mahony, M.P., 2014) that utilises the extracted entities from phase 1 to recommend the most appropriate resumes for the given job description. The system employs concepts like Vectorisation (Salton, G., Wong, A. and Yang, C.S., 1975), importance or weight assigning techniques like TF-IDF (Jabri, Siham, et al., 2018) and similarity measures like cosine distance (Huang and Anna, 2008) for calculating the similarity among the contents of the documents.

**7.2.2 TF-IDF**

TF-IDF stands for “Term Frequency – Inverse Document Frequency” (Stecanella, 2020). The TF-IDF weight is often used in text mining techniques. TF-IDF was invented for information retrieval and document search. This weight is a numerical measure to determine how important a term is with respect to a document in a collection or corpus. The importance increases proportionally to the frequency of a word within the document but is offset by the number of documents that contain the word. So, terms that are frequently used in every document, such as this, and, what, whom, is, the, if, etc. rank low even though they may appear many times since they don’t mean much to that document in particular (Stecanella, 2020). The TF-IDF value for a term in a document is calculated by multiplying two different metrics (Stecanella, 2020) as shown in equation (1) below

𝑇𝐹 − 𝐼𝐷𝐹 (𝑡, 𝑑) = 𝑇𝐹 (𝑡, 𝑑) ∗ 𝐼𝐷𝐹 (𝑡, 𝑑)

Term Frequency: It measures how frequently a word occurs in each document in the corpus. Since a word may occur more number of times in lengthy documents than shorter ones, so you need to adjust or normalize this frequency. A normalized term frequency is calculated by dividing the number of times a term appears in a document by the total number of terms in that document. Mathematically, we can write it as (Jabri, Siham, et al., 2018) shown below in equation (2).

𝑇𝐹 (𝑡, 𝑑) = 𝑓𝑟𝑒𝑞 (𝑡, 𝑑) / ∑ 𝑓𝑟𝑒𝑞 (𝑡𝑖 , 𝑑)

Here, freq (t, d) is the count of the instances of the term t in document d, TF (t, d) is the proportion of the count of term t in document d, and n is the number of distinct terms in document d.

Inverse Document Frequency: It measures how important a word is for all documents in the corpus. In other words, this metric helps to know how rare or common a word is across in the corpus. It weighs down the terms that occur more often while scaling up the rare terms. The terms that appear more often in the set of documents have IDF value close to 0 while the rare terms have a high IDF. It is calculated by dividing the total number of documents by the number of documents that contain a term and then calculating the logarithm (Stecanella, 2020). Mathematically, we can write it as shown below in equation (3).

𝐼𝐷𝐹(𝑡) = 𝑙𝑜𝑔 ( 𝑁 / 𝑐𝑜𝑢𝑛𝑡(𝑡) )

Here, N is the number of distinct documents in the corpus and count (t) is the number of documents in the corpus in which the term t is present. The product of these two metrics i.e. equation (2) and (3) results in a TFIDF score of a word in a document. More relevance of a word in a document is reflected by its high TF-IDF score. In our system, we modelled the resumes and the job description document into a vector space. This is done by creating a dictionary of terms present in the documents and converting each term to a dimension in the vector space. We then computed the TFIDF matrix for the CVs and the job query by using the CountVectorizer and the TfidfTransformer python modules. In the next step, we need to calculate the similarity score between the resumes and the job description.

**7.2.3 Cosine similarity**

A Similarity measure is a metric that determines how much the two objects are alike. Cosine similarity (Sidorov, Grigori, et al., 2014) is a measure to find how similar the two documents are regardless of their size. It represents the orientation of the documents when plotted on an Ndimensional space, where each dimension depicts the features of the object. It’s a symmetrical algorithm, which implies that the results from computing the similarity of item X to item Y is equal to computing the similarity of item Y to item X. Mathematically, we can represent it as shown below (Sidorov, Grigori, et al., 2014) in equation (4)

𝑐𝑜𝑠(𝜃) = 𝑎 ⇀ . 𝑏 ⇀ ||𝑎 → || ||𝑏 → || = ∑ 𝑎𝑖𝑏𝑖 𝑛 𝑖=1 √∑ 𝑎𝑖 2 𝑛 𝑖=1 √∑ 𝑏i2

Here, 𝐚. ⇀ 𝐛 ⇀ = ∑ 𝑎𝑖𝑏𝑖 𝑛 1 = 𝑎1𝑏1 + 𝑎2𝑏2+ . . . +𝑎𝑛𝑏𝑛 is the dot product of the two vectors. Using this formula, we calculate the cosine similarity between all pairs of elements. It can then be used to rank the resume documents with respect to a given vector of query words. However, cosine similarity focuses on features that are related to the text’s words only and will give less accurate results. The efficiency of similarity measures can be improved by the inclusion of semantic information. This will constitute the future scope for our automated resume screening system.

# CHAPTER - 8

# IMPLEMENTATION

## Implementation Tools:

### Programming Language and Coding Tools:

System implementation includes the usage of tools for the development of the project. This project uses Machine Learning technology with Python as programming language and Anaconda distribution and implementation of coding is done in the Jupyter notebook.

### Python

Python is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language), [high-level](https://en.wikipedia.org/wiki/High-level_programming_language), [general-purpose](https://en.wikipedia.org/wiki/General-purpose_programming_language) [programming language.](https://en.wikipedia.org/wiki/Programming_language) Created by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) and first released in 1991, Python's design philosophy emphasizes [code](https://en.wikipedia.org/wiki/Code_readability) [readability](https://en.wikipedia.org/wiki/Code_readability) with its notable use of [significant whitespace.](https://en.wikipedia.org/wiki/Off-side_rule) Its language constructs and [object-](https://en.wikipedia.org/wiki/Object-oriented_programming) [oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is [dynamically typed](https://en.wikipedia.org/wiki/Dynamic_programming_language) and [garbage-collected.](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) It supports multiple [programming](https://en.wikipedia.org/wiki/Programming_paradigm) [paradigms,](https://en.wikipedia.org/wiki/Programming_paradigm) including [procedural,](https://en.wikipedia.org/wiki/Procedural_programming) object-oriented, and [functional programming](https://en.wikipedia.org/wiki/Functional_programming). Python is often described as a "batteries included" language due to its comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

Python was conceived in the late 1980s as a successor to the [ABC language](https://en.wikipedia.org/wiki/ABC_(programming_language)). system capable of collecting [reference cycles](https://en.wikipedia.org/wiki/Reference_cycle). Python 3.0, released 2008, was a major revision of the language that is not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility), and much Python 2.0, released 2000, introduced features like [list comprehensions](https://en.wikipedia.org/wiki/List_comprehension) and a [garbage collection](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) on 2 code does not run unmodified on Python 3. Due to concern about the amount of code written for Python 2, support for Python 2.7 (the last release in the 2.x series) was extended to 2020. Language developer Guido van Rossum shouldered sole responsibility for the project until July 2018 but now shares his leadership as a member of a five-person steering council.

Python [interpreters](https://en.wikipedia.org/wiki/Interpreter_(computing)) are available for many [operating systems.](https://en.wikipedia.org/wiki/Operating_system) A global community of programmers develops and maintains [CPython](https://en.wikipedia.org/wiki/CPython), an [open source](https://en.wikipedia.org/wiki/Open-source_software) [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation). A [non-profit organization,](https://en.wikipedia.org/wiki/Nonprofit_organization) the [Python Software Foundation](https://en.wikipedia.org/wiki/Python_Software_Foundation), manages and directs resources for Python and CPython development.

### Anaconda

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution includes data-science packages suitable for Windows, Linux, and macOS. Anaconda distribution comes with 1,500 packages selected from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command-line interface (CLI).

Open-source packages can be individually installed from the Anaconda repository, Anaconda Cloud (anaconda.org), or your private repository or mirror, using the conda install command. Anaconda Inc compiles and builds the packages available in the Anaconda repository itself, and provides binaries for Windows 32/64 bit, Linux 64 bit and MacOS 64-bit. Anything available on PyPI may be installed into a conda environment using pip, and conda will keep track of what it has installed itself and what pip has installed. Custom packages can be made using the conda build command and can be shared with others by uploading them to Anaconda Cloud, PyPI, or other repositories. The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, it is possible to create new environments that include any version of Python packaged with conda.

### Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments, and channels without using command-line commands. Navigators can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS, and Linux.

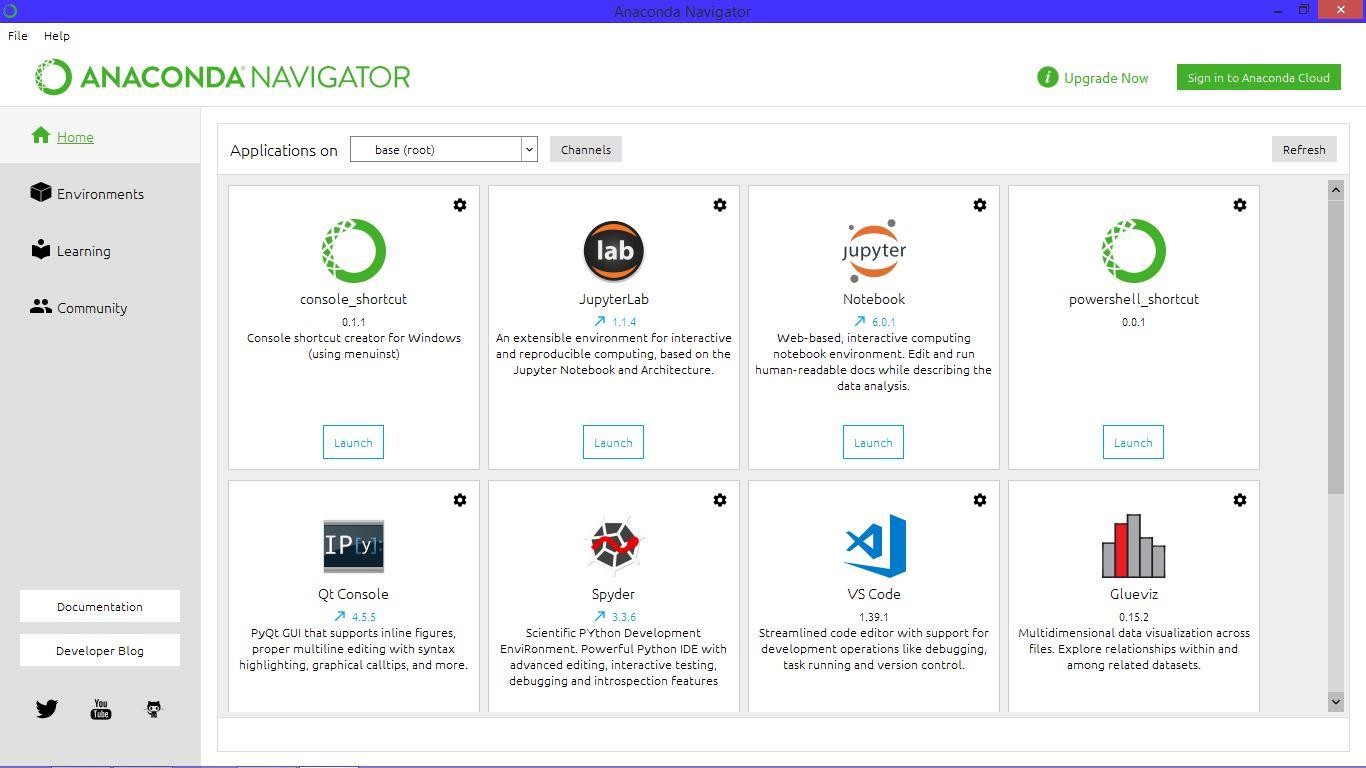


Fig 8.1 Anaconda Navigator

### Jupyter Notebook

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents—which are a type of computational notebook. The "notebook" term can colloquially refer to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells that can contain code, text (using Markdown), mathematics, plots, and rich media, usually ending with the “ipynb”, extension.

Jupyter Notebook and its flexible interface extends the notebook beyond code to visualization, multimedia, collaboration, and more. In addition to running your code, it stores code and output, together with markdown notes, in an editable document called a notebook. When you save it, this is sent from your browser to the notebook server, which saves it on disk as a JSON file with a .ipynb extension.

Jupyter Notebook can connect to many kernels to allow programming in many languages. By default, Jupyter Notebook ships with the IPython kernel. The notebook server, not the kernel, is responsible for saving and loading notebooks, so you can edit notebooks even if you don’t have the kernel for that language—you just won’t be able to run code. The kernel doesn’t know anything about the notebook document: it just gets sent cells of code to execute when the user runs them. As of the 2.3 release (October 2014), there are currently 49 Jupyter-compatible kernels for many programming languages, including Python, R, Julia, and Haskell.

The Notebook interface was added to IPython in the 0.12 release (December 2011), renamed to Jupyter notebook in 2015 (IPython 4.0 – Jupyter 1.0). Jupyter Notebook is similar to the notebook interface of other programs such as Maple, Mathematica, and SageMath, a computational interface style that originated with Mathematica in the 1980s. According to The Atlantic, Jupyter interest overtook the popularity of the Mathematical notebook interface in early 2018.

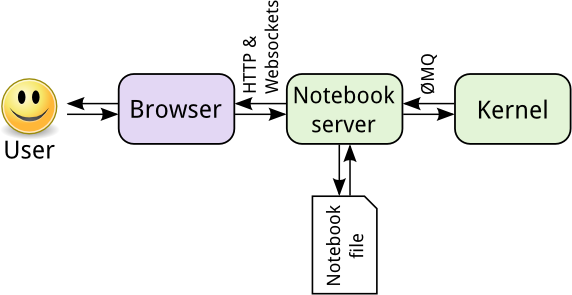


Fig 8.2 Jupyter Notebook Interface

**CHAPTER – 9**

**CODING**

## 1. EDA

In [ ]:

import pandas as pd

import numpy as np

In [ ]:

data\_raw = pd.read\_csv("train.csv")

data\_raw.shape

In [ ]:

print("Number of rows in data =",data\_raw.shape[0])

print("Number of columns in data =",data\_raw.shape[1])

print("\n")

data\_raw.head()

### 1.1. Checking for missing values

In [ ]:

missing\_values\_check = data\_raw.isnull().sum()

print(missing\_values\_check)

### 1.2. Calculating number of comments under each label

In [ ]:

rowSums = data\_raw.iloc[:,2:].sum(axis=1)

clean\_comments\_count = (rowSums==0).sum(axis=0)

print("Total number of comments = ",len(data\_raw))

print("Number of clean comments = ",clean\_comments\_count)

print("Number of comments with labels =",(len(data\_raw)-clean\_comments\_count))

In [ ]:

categories = list(data\_raw.columns.values)

categories = categories[2:]

print(categories)

In [ ]:

counts = []

for category in categories:

counts.append((category, data\_raw[category].sum()))

df\_stats = pd.DataFrame(counts, columns=['category', 'number of comments'])

df\_stats

## 2. Data Pre-Processing

In [ ]:

data = data\_raw

data = data\_raw.loc[np.random.choice(data\_raw.index, size=2000)]

data.shape

In [ ]:

import nltk

from nltk.corpus import stopwords

from nltk.stem.snowball import SnowballStemmer

import re

import sys

import warnings

if not sys.warnoptions:

warnings.simplefilter("ignore")

### 2.1. Cleaning Data

In [ ]:

def cleanHtml(sentence):

cleanr = re.compile('<.\*?>')

cleantext = re.sub(cleanr, ' ', str(sentence))

return cleantext

def cleanPunc(sentence): *#function to clean the word of any punctuation or special characters*

cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)

cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)

cleaned = cleaned.strip()

cleaned = cleaned.replace("\n"," ")

return cleaned

def keepAlpha(sentence):

alpha\_sent = ""

for word in sentence.split():

alpha\_word = re.sub('[^a-z A-Z]+', ' ', word)

alpha\_sent += alpha\_word

alpha\_sent += " "

alpha\_sent = alpha\_sent.strip()

return alpha\_sent

In [ ]:

data['comment\_text'] = data['comment\_text'].str.lower()

data['comment\_text'] = data['comment\_text'].apply(cleanHtml)

data['comment\_text'] = data['comment\_text'].apply(cleanPunc)

data['comment\_text'] = data['comment\_text'].apply(keepAlpha)

data.head()

### 2.2. Removing Stop Words

In [ ]:

import nltk

nltk.download('stopwords')

In [ ]:

stop\_words = set(stopwords.words('english'))

stop\_words.update(['zero','one','two','three','four','five','six','seven','eight','nine','ten','may','also','across','among','beside','however','yet','within'])

re\_stop\_words = re.compile(r"\b(" + "|".join(stop\_words) + ")\\W", re.I)

def removeStopWords(sentence):

global re\_stop\_words

return re\_stop\_words.sub(" ", sentence)

data['comment\_text'] = data['comment\_text'].apply(removeStopWords)

data.head()

### 2.3 Stemming

In [ ]:

stemmer = SnowballStemmer("english")

def stemming(sentence):

stemSentence = ""

for word in sentence.split():

stem = stemmer.stem(word)

stemSentence += stem

stemSentence += " "

stemSentence = stemSentence.strip()

return stemSentence

data['comment\_text'] = data['comment\_text'].apply(stemming)

data.head()

### 2.4 Train-Test Split

In [ ]:

from sklearn.model\_selection import train\_test\_split

train, test = train\_test\_split(data, random\_state=42, test\_size=0.30, shuffle=True)

print(train.shape)

print(test.shape)

In [ ]:

train\_text = train['comment\_text']

test\_text = test['comment\_text']

### 2.5 TF-IDF

In [ ]:

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(strip\_accents='unicode', analyzer='word', ngram\_range=(1,3), norm='l2')

vectorizer.fit(train\_text)

vectorizer.fit(test\_text)

In [ ]:

x\_train = vectorizer.transform(train\_text)

y\_train = train.drop(labels = ['id','comment\_text'], axis=1)

x\_test = vectorizer.transform(test\_text)

y\_test = test.drop(labels = ['id','comment\_text'], axis=1)

## 3. Multi-Label Classification

In [ ]:

%%time

from sklearn.svm import SVR

from sklearn.pipeline import Pipeline

from sklearn.metrics import accuracy\_score

from sklearn.multiclass import OneVsRestClassifier

svmReg\_pipeline = Pipeline([

('clf', OneVsRestClassifier(SVR(kernel = 'rbf'), n\_jobs=-1)),

])

for category in categories:

print('\*\*Processing {} comments...\*\*'.format(category))

*# Training logistic regression model on train data*

svmReg\_pipeline.fit(x\_train, train[category])

*# calculating test accuracy*

prediction = svmReg\_pipeline.predict(x\_test)

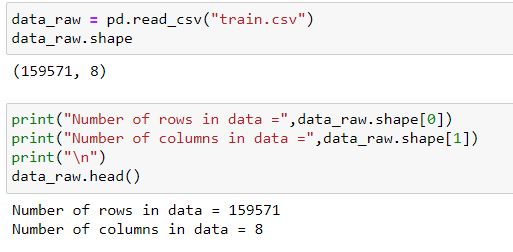
print('Test accuracy is {}'.format(accuracy\_score(test[category], prediction)))

print("\n")

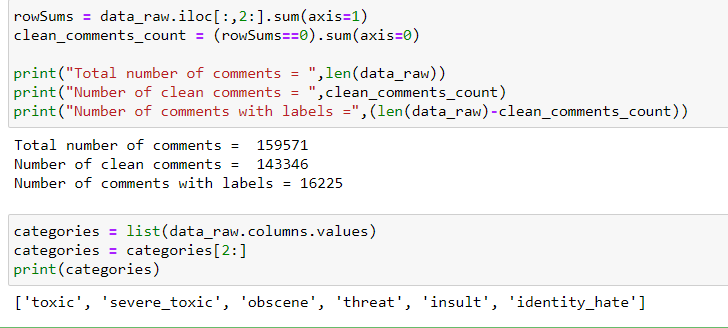
**CHAPTER - 10**

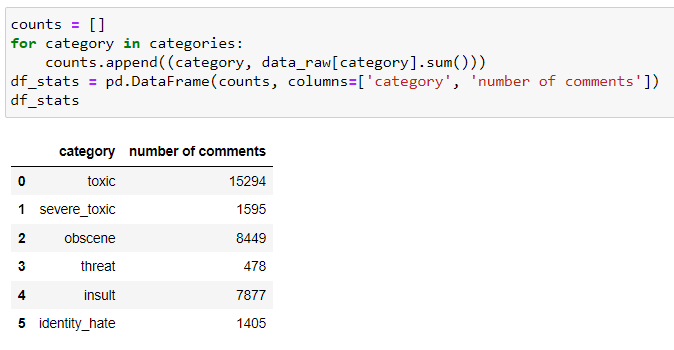
# OUTPUTS

# 10.1 Download Dataset & Importing Dataset

****

# Calculating number of comments under each label





# 10.3 Removing Stop Words



# 10.4 Stemming

# 10.5 Multi-Label Classification using SVM

# 

# 10.6 Processing outputs for every category comments with accuracy

# 

# CONCLUSION

It is important to make sure that people with different ideas are heard without the fear of any toxic and hateful remarks. And after analyzing using npl and SVM approaches to solve this problem of classification of toxic comments online, it is found that the accuracy of toxic comments 92.66%, severe toxic comments 98.66%, obscene comments 95.33%, Threat comments 99.83%, insult comments 96.16% and Threat comments 99.83%, insult comments 99.66%.

# FUTURE SCOPE

Future scope for this analysis would be integrating such classification algorithms into social media platforms to automatically classify and censor or toxic comments.

# BIBLIOGRAPHY

[1] “Toxic Comment Classification using Natural Language Processing”, A. Akshith Sagar, J. Sai Kiran, 2020, IRJET

[2]Sida Wang and Christopher D. Manning, “Baselines and Bigrams: Simple, Good Sentiment and Topic Classification”, Stanford, CA,

[3] Mujahed A. Saif, Alexander N. Medvedev, Maxim A. Medvedev, and Todorka Atanasova, “Classification of online toxic comments using the logistic regression and neural networks models”, AIP Conference Proceedings 2048, 060011 (2018)

[4] https://www.data-to-viz.com/

[5] https://www.kaggle.com/jagangupt a/stop-the-s-toxiccomments-eda