MACHINE LEARNING

NLP – Review Classification

Vaishnavi Madisetti -NGIT

Sri Harika Kukkadapu- NGIT

Janumpally Vamshi Krishna-TKRCET

Vasa Sai Priya-NGIT

Group Number-14

Project Number-3

ABSTRACT

This project deals with predicting good and bad reviews given by costumer in restaurant using Natural Language Processing and Machine Learning algorithms. Sentiment analysis gained huge popularity with increase in usage of social media. It is also used in many other applications like customer service. This can significantly increase profits for restaurant. Classifying reviews manually can become typical if there are a greater number of reviews. This approach of sentiment analysis train our model with previous reviews and predict the type of review for new reviews. Customer service is prominent feature in any business; hence sentiment analysis gives better satisfaction and leads to good revenues for a company. The process of sentiment analysis involves not only ML and NLP algorithms but also some linguistic approach. When it comes to classify restaurant reviews it involves positive, negative and sarcastic reviews (giving negative reviews using positive words), hence this phase of become more problematic. Sentiment analysis, the automated extraction of expressions of positive or negative attitudes from text. The tools provided by natural language processing and machine learning along with other approaches to work with large volumes of text, makes it possible to begin extracting sentiments from given text. Refining the text (removing symbols) and removing stop words (unwanted words) makes this process easier. After that we encode the refined data and create a model and train it.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

1.1 MACHINE LEARNING:

Machine learning (ML) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop conventional algorithms to perform the needed tasks.

Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

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 algorithm is an evolution of the regular algorithm. It makes your programs “smarter”, by allowing them to automatically learn from the data you provide. The algorithm is mainly divided into:

- Training Phase
- Testing phase

So, building upon the example I had given a while ago, let’s talk a little about these phases.

Training Phase

You take a randomly selected specimen of apples from the market (training data), make a table of all the physical characteristics of each apple, like colour, size, shape, grown in which part of the country, sold by which vendor, etc (features), along with the sweetness, juiciness, ripeness of that apple (output variables). You feed this data to the machine learning algorithm (classification/regression), and it learns a model of the correlation between an average apple’s physical characteristics, and its quality.

Testing Phase

Next time when you go shopping, you will measure the characteristics of the apples which you are purchasing (test data) and feed it to the Machine Learning algorithm. It will use the model which was computed earlier to predict if the apples are sweet, ripe and/or juicy. The algorithm may internally use the rules, similar to the one you manually wrote earlier (for e.g., a decision tree). Finally, you can now shop for apples with great confidence, without worrying about the details of how to choose the best apples.

1.1.1 MACHINE LEARNING METHODS:

Machine learning algorithms are often categorized as supervised or unsupervised.

1. Supervised machine learning algorithms can apply what has been learned in the past to new data using labelled examples to predict future events. Starting from the analysis of a known training

dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

2. In contrast, unsupervised machine learning algorithms are used when the information used to train is neither classified nor labelled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabelled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabelled data.

3. Semi-supervised machine learning algorithms fall somewhere in between supervised and unsupervised learning, since they use both labelled and unlabelled data for training – typically a small amount of labelled data and a large amount of unlabelled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labelled data requires skilled and relevant resources in order to train it / learn from it. Otherwise, acquiring unlabelled data generally doesn't require additional resources.

4. Reinforcement machine learning algorithms is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behaviour within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.

1.1.2 MACHINE LEARNING ALGORITHMS:

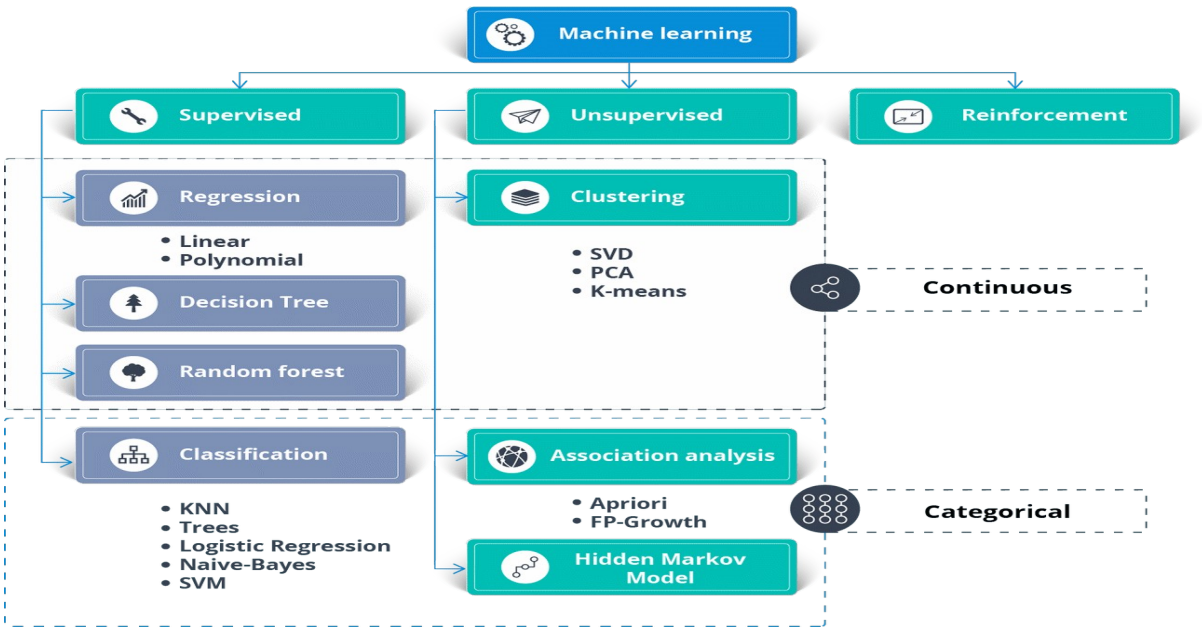


Fig1.1: Classification of algorithms

SUPERVISED MACHINE LEARNING ALGORITHMS

Here is the list of most commonly used machine learning algorithms.

1. Linear Regression
2. Multi Linear Regression

3. Logistic Regression
4. Decision Tree
5. Random Forest
6. Naive Bayes
7. KNN
8. Support Vector Machine (SVM)
9. SVM with PCA

In this project we have discussed about SVM, KNN, SVM with PCA.

1.Support Vector Machine:

Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, that is, the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

1) Hyperplanes and Support Vectors:

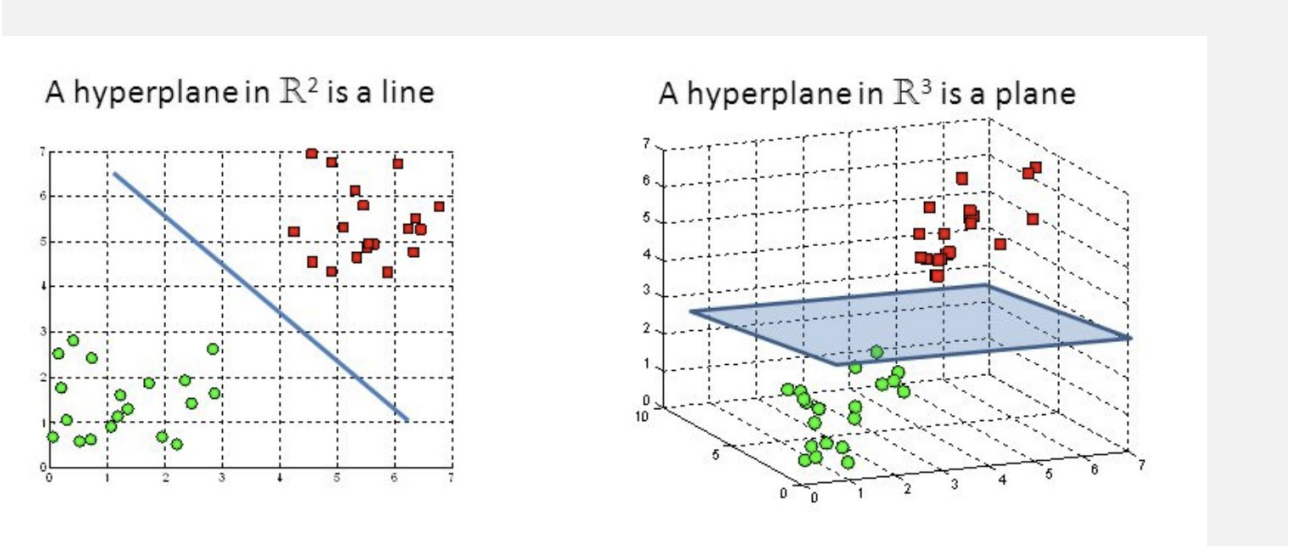


Fig1.2: Hyperplanes in 2D and 3D feature space

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.

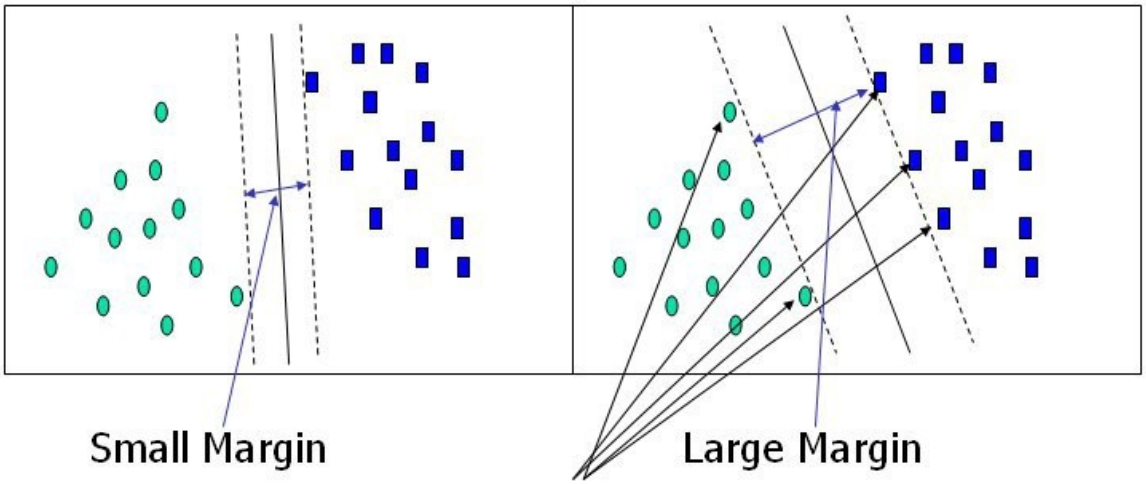


Fig 1.3:

2)Support Vectors:

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

There is a simple way to implement the SVM algorithm. We can use the Scikit learn library and just call the related functions to implement the SVM model.

3) Tuning parameters: Kernel, Regularization:

Kernel:

The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role.

For linear kernel the equation for prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

f(x) = B(0) + sum(ai * (x,xi))

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 B0 and ai (for each input) must be estimated from the training data by the learning algorithm.

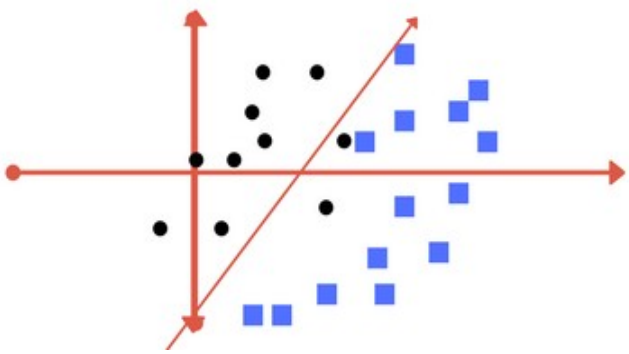
The polynomial kernel can be written as $K(x,xi) = 1 + sum(x * xi)^d$ and exponential as $K(x,xi) = exp(-gamma * sum((x — xi^2))$.

Regularization:

The Regularization parameter (often termed as C parameter in python's sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example.

For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

The images below are example of two different regularization parameter. Left one has some misclassification due to lower regularization value. Higher value leads to results like right one.



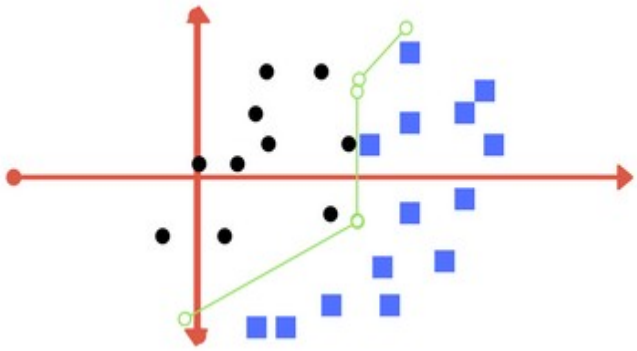


Fig 1.4: Parameter Regularization

2.K-Nearest Neighbour (KNN)

· K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

· K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

· K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

· K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

· K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

· It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

· KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

1) **Need for KNN**
Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

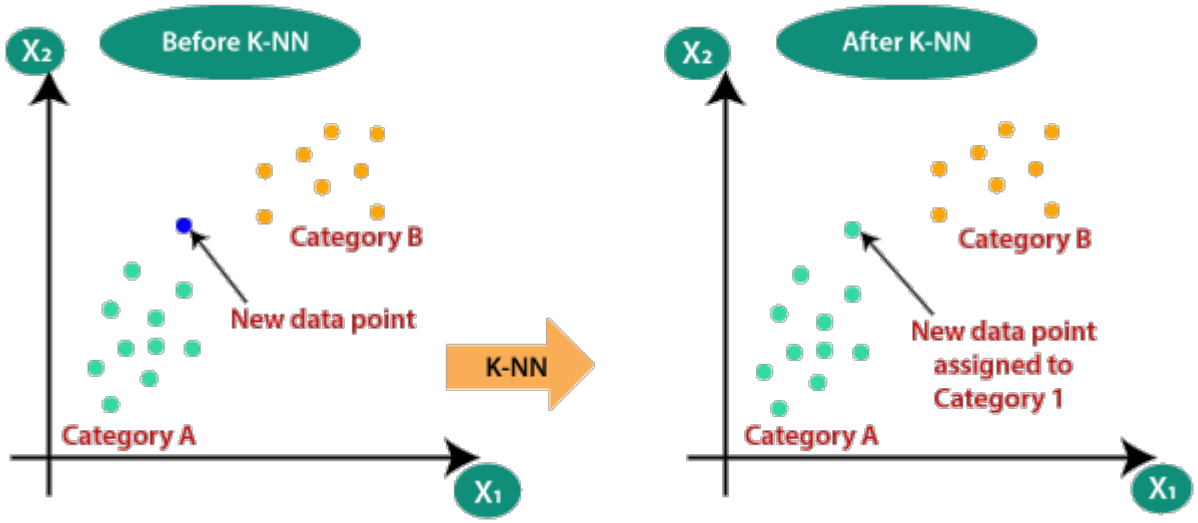


Fig 1.5: KNN

2) How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

· Step-1: Select the number K of the neighbors

· Step-2: Calculate the Euclidean distance of K number of neighbors

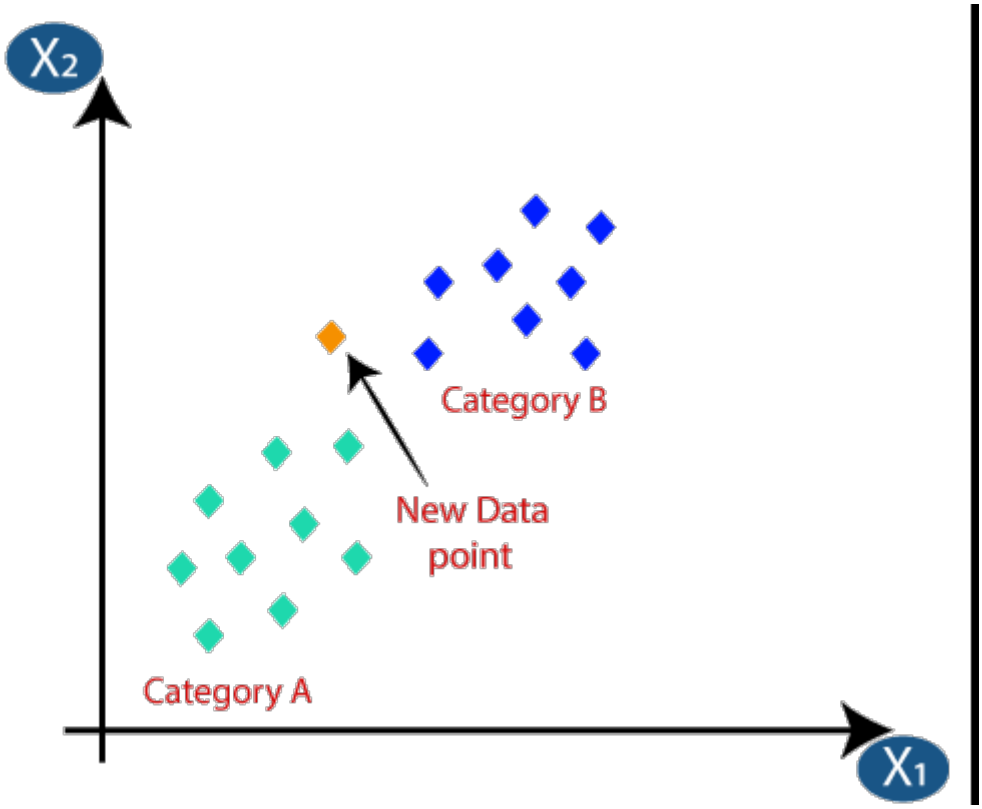
· Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

· Step-4: Among these k neighbors, count the number of the data points in each category.

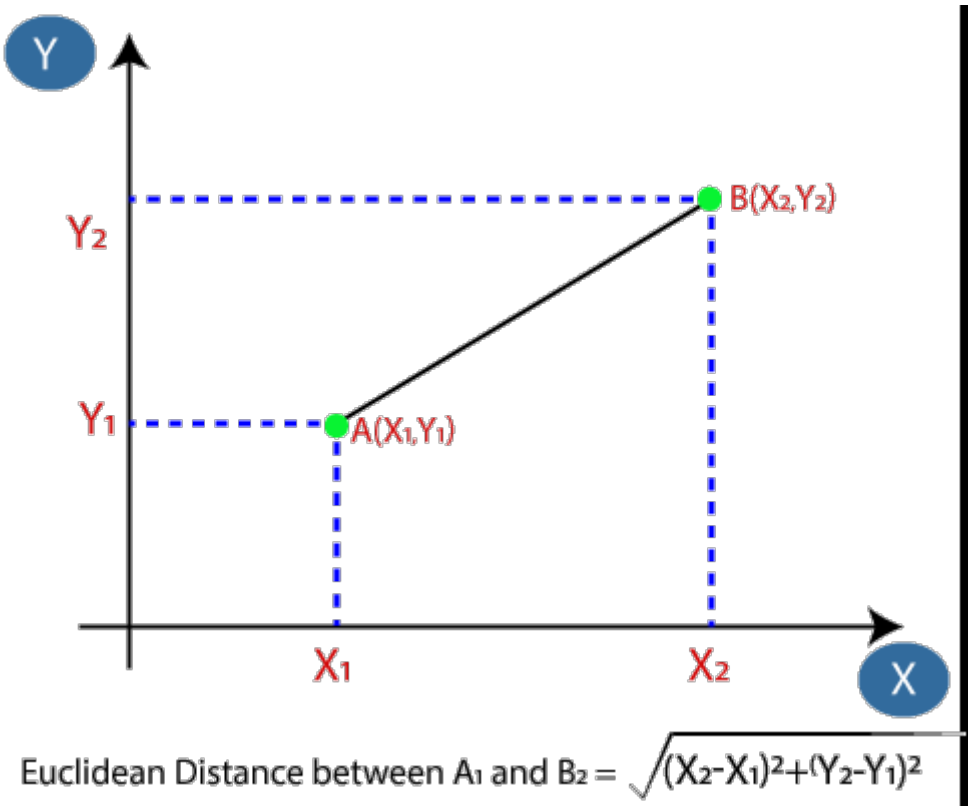
· Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

· Step-6: Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



- By calculating the Euclidean distance, we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:

o As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

3)Advantages and Disadvantages:

Advantages
1.The algorithm is simple and easy to implement.
2.There's no need to build a model, tune several parameters, or make additional assumptions.
3.The algorithm is versatile. It can be used for classification, regression, and search (as we will see in the next section).
Disadvantages
1.The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

4) selecting the value of K in the K-NN Algorithm

Below are some points to remember while selecting the value of K in the K-NN algorithm:
o There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
o A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
o Large values for K are good, but it may find some difficulties.

3.SVM With PCA:
(Principal Component Analysis)

PCA is:
• Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.
• Unsupervised Machine Learning
• A transformation of your data and attempts to find out what features explain the most variance in your data. For example:

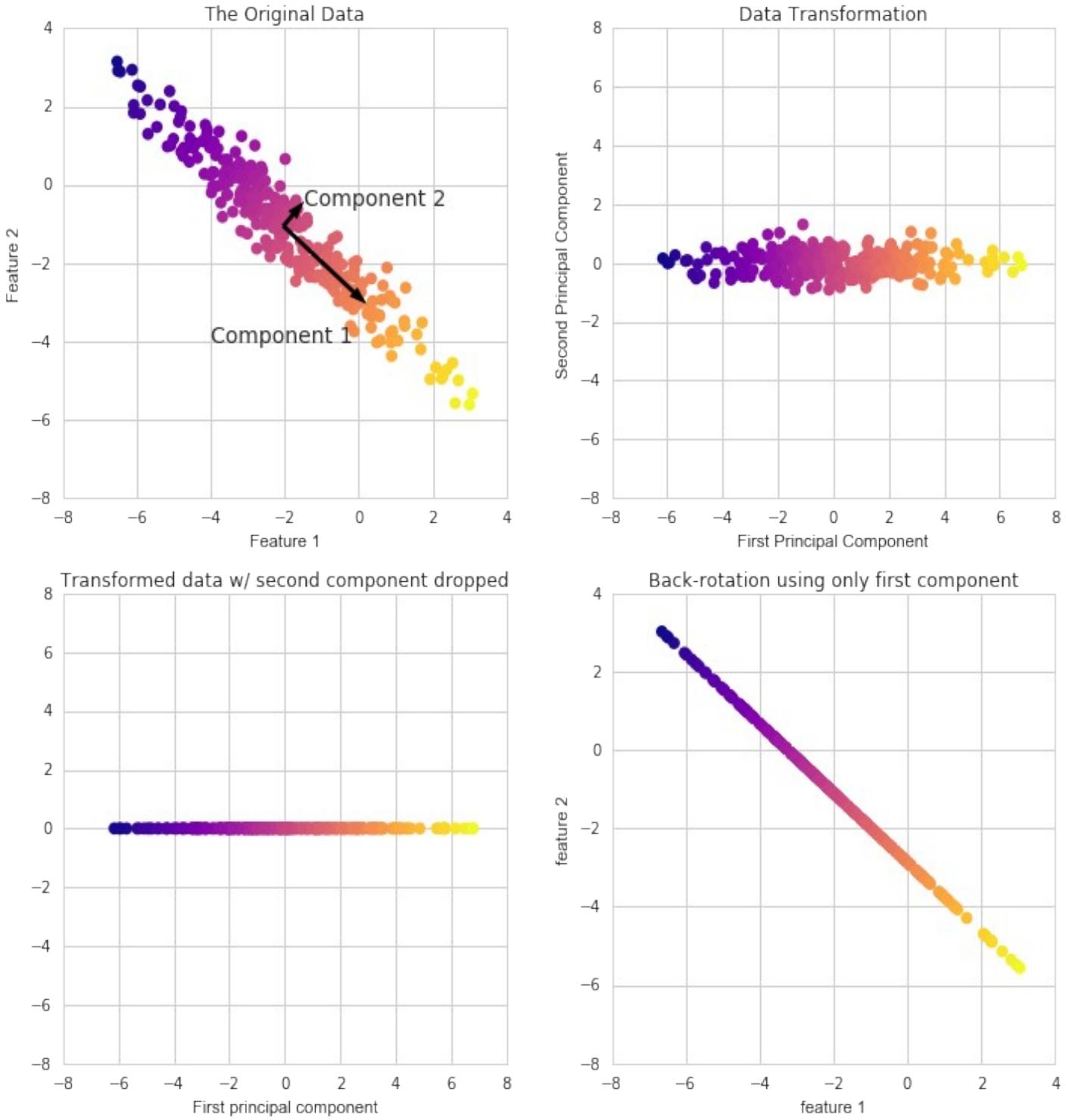


Fig 1.6: SVM with PCA (Features)

PCA with Scikit Learn uses a very similar process to other preprocessing functions that come with SciKit Learn. We instantiate a PCA object, find the principal components using the fit method, then apply the rotation and dimensionality reduction by calling transform ().
We can also specify how many components we want to keep when creating the PCA object.

CLASSICAL PCA FORMULATION

0. Standardize the raw data: make the raw data have zero mean and unit variance

$$x_j^{(i)} = \frac{x_j^{(i)} - \bar{x}_j}{\sigma_j} \quad \forall j \tag{1}$$

1. Get the covariance matrix of raw data, namely:

$$\Sigma = \frac{1}{m}\sum_i^m (x_{(i)})(x_{(i)})^T, \quad \Sigma \in \mathbb{R}^{n\times n} \tag{2}$$

2. Compute the eigenvalue and eigenvector of the covariance matrix:

$$u^T \Sigma = \lambda u \tag{3}$$

$$U = \left\{ \begin{array}{cccc} | & | & & | \\ u_1 & u_2 & \dots & u_n \\ | & | & & | \end{array} \right\}, \quad u_i \in \mathbb{R}^n \tag{4}$$

   Here, eigenvector is corresponding to its eigenvalue, and they are placed in the descending order of eigenvalue.

3. Project the raw data into a k-dimensional subspace: Choose the top k eigenvector of covariance matrix. The top k eigenvectors now form a new, orthogonal basis for the data. Then to represent raw data in the new basis, we need only compute the corresponding vector as shown below

$$x_{new}^{(i)} = \begin{bmatrix} u_1^T x^{(i)} \\ u_2^T x^{(i)} \\ \dots \\ u_k^T x^{(i)} \end{bmatrix} \in \mathbb{R}^k \tag{5}$$

   Thus, whereas raw data is with n dimensionality, and the new vector new gives a lower, k-dimensional, approximation/representation for the raw data.

Principal Component Analysis:
- Used in exploratory data analysis (EDA)
- Visualize genetic distance and relatedness between populations.
- Method:
  - Eigenvalue decomposition of a data covariance (or correlation) matrix
  - Singular value decomposition of a data matrix (After mean centering / normalizing) the data matrix for each attribute.
- Output
  - Component scores, sometimes called **factor scores** (the transformed variable values)
  - **loadings** (the weight)
- Data compression and information preservation
- Visualization
- Noise filtering
- Feature extraction and engineering

CHAPTER 2

NATURAL LANGUAGE PROCESSING

2. NATURAL LANGUAGE PROCESSING

Natural Language Processing

Natural Language Processing (NLP) is a branch of artificial intelligence that helps computers read, interpret, and understand human language. The goal of NLP is for machines to carry out repetitive and high-volume tasks that would otherwise be completed by humans.

Once a fantasy of science fiction movies, the ability of machines to interpret human language is now at the core of many applications that we use every day – from translation software, chatbots, spam filters, and search engines, to grammar checking software, voice assistants, and social media monitoring tools.

NPL is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.

Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural language generation.

2.1 What Is Natural Language Processing (NLP)?

Natural Language Processing (NLP) is a field of Artificial Intelligence (AI) that focuses on quantifying human language to make it intelligible to machines. It combines the power of linguistics and computer science to study the rules and structure of language, and create intelligent systems capable of understanding, analyzing, and extracting meaning from text and speech.

Linguistics is used to understand the structure and meaning of a text by analyzing different aspects like syntax, semantics, pragmatics, and morphology. Then, computer science transforms this linguistic knowledge into rule-based or machine learning algorithms that can solve specific problems and perform desired tasks.
Take Gmail, for example. Your emails are automatically categorized as Promotions, Social, Primary, *or* Spam thanks to an NLP task called text classification. By breaking down words and identifying patterns, rules, and relationships between them, machines automatically learn which category to assign emails.

2.2 Why Is Natural Language Processing Important?

Natural Language Processing plays a very important role in structuring big data because it prepares text and speech for machines so that they're able to interpret, process, and organize information. Some of the main advantages of NLP include:

- **Large-scale analysis.** Natural Language Processing can help machines perform language-based tasks such as reading text, identifying what's important, and detecting sentiment at scale. If you receive an influx of customer support tickets, you don't need to hire more staff. NLP tools can be scaled up or down as needed.

- **Automate processes in real-time.** Machine learning tools, equipped with natural language processing, can learn to understand and analyze information without human help – quickly, effectively, and around the clock.
- **Consistent and unbiased criteria.** NLP machines are not subjective like humans. They tag data based on one set of rules, so you don't have to worry about inconsistent and inaccurate results.

2.3 Techniques of Natural Language Processing & Algorithms

In this section, we'll focus on two primary natural language processing techniques and their sub-tasks.

Syntactic Analysis

Syntactic analysis — also known as parsing or syntax analysis — identifies the syntactic structure of a text and the dependency relationships between words, represented on a diagram called a parse tree.

Syntax analysis involves many different sub-tasks, including:

Tokenization
This is the most basic task in natural language processing. It's used to break up a string of words into semantically useful units called tokens and works by defining boundaries, that is, a criterion of where a token begins or ends.

You can use sentence tokenization to split sentences within a text, or word tokenization to split words within a sentence. Generally, word tokens can be separated by blank spaces, and sentence tokens by stops. However, you can perform high-level tokenization for more complex structures, like words that often go together, otherwise known as collocations (for example, New York).

Here's an example of how word tokenization simplifies text:

Customer service couldn't be better! = ["customer service", "could", "not", "be", "better"]

Part-of-speech tagging
Part-of-speech tagging (abbreviated as PoS tagging) involves adding a part of speech category to each token within a text. Some common PoS tags
are verb, adjective, noun, pronoun, conjunction, preposition, intersection, among others. In this case, the example above would look like this:
"Customer service": NOUN, "could": VERB, "not": ADVERB, be": VERB, "better": ADJECTIVE, "!": PUNCTUATION
PoS tagging is useful for identifying relationships between words and, therefore, understand the meaning of sentences.

Dependency Parsing
Dependency grammar refers to the way the words in a sentence are connected to each other. A dependency parser, therefore, analyzes how 'head words' are related and modified by other words in order to understand the syntactic structure of a sentence:

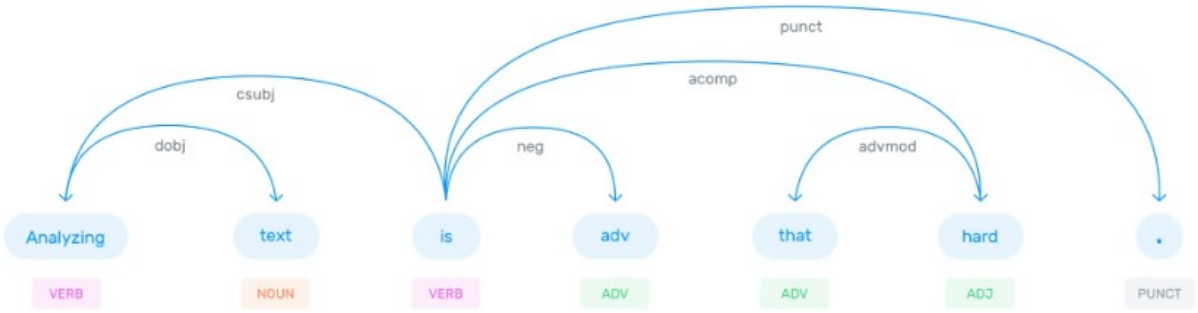


Fig 2.1: Dependency Parsing

Constituency Parsing

Constituency Parsing aims to visualize the entire syntactic structure of a sentence by identifying phrase structure grammar. Basically, it consists of using abstract terminal and non-terminal nodes associated to words, as shown in this example:

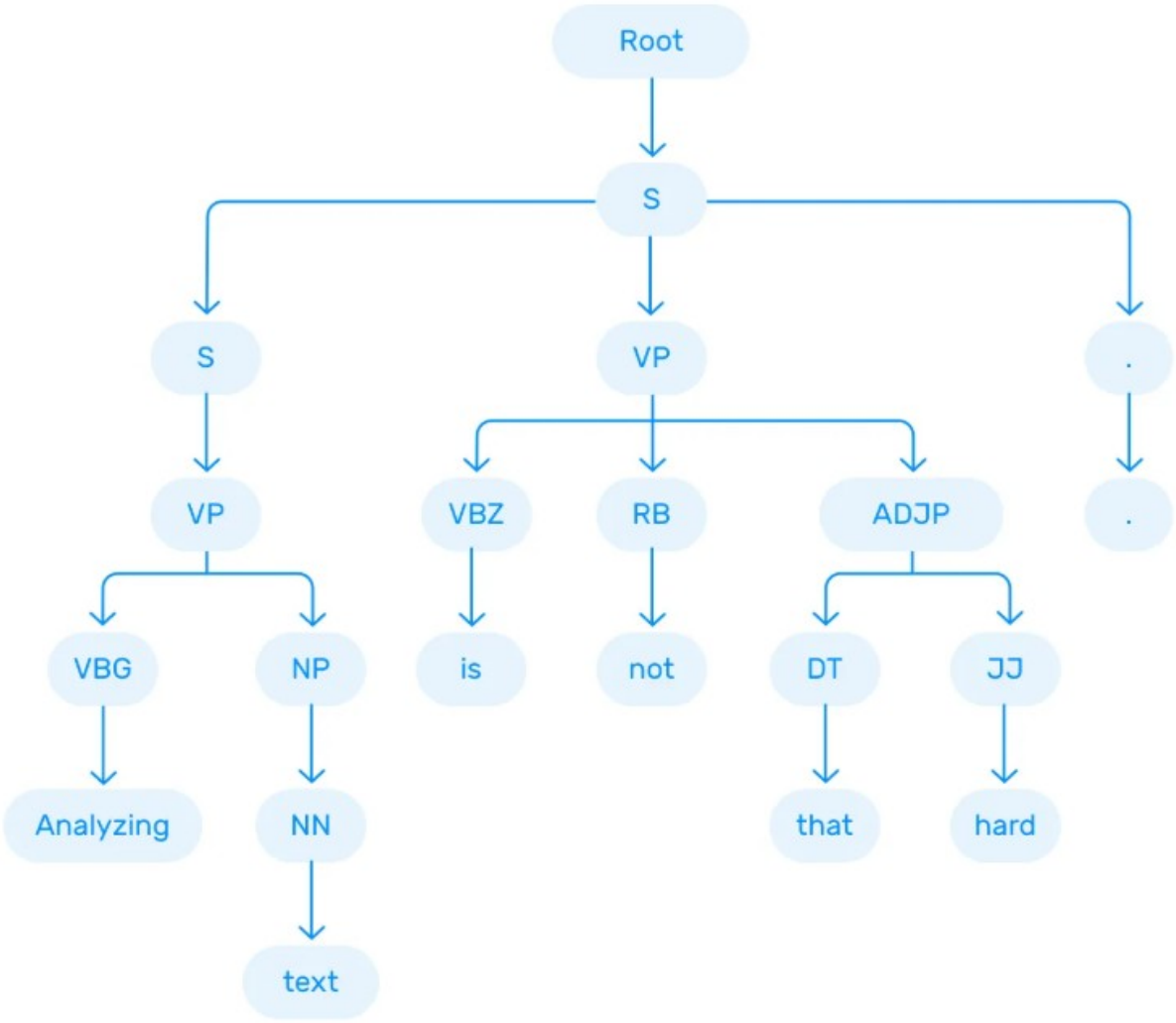


Fig 2.2: Constituency Parsing

You can try different parsing algorithms and strategies depending on the nature of the text you intend to analyze, and the level of complexity you'd like to achieve.

Lemmatization & Stemming

When we speak or write, we tend to use inflected forms of a word (words in their different grammatical forms). To make these words easier for computers to understand, NLP uses lemmatization and stemming to transform them back to their root form.

The word as it appears in the dictionary – its root form – is called a lemma. For example, the words 'are, is, am, were, and been', are grouped under the lemma 'be'. So, if we apply this lemmatization to "African elephants have 4 nails on their front feet", the result will look something like this:
African elephants have 4 nails on their front feet = ["African", "elephant", "have", "4", "nail", "on", "their", "foot"]
This example is useful to see how the lemmatization changes the sentence using its base form (e.g. the word "feet" was changed to "foot).

When we refer to stemming, the root form of a word is called a stem. Stemming 'trims' words, so word stems may not always be semantically correct.

For example, stemming the words "consult", "consultant", "consulting", and "consultants", would result in the root form "consult".

While lemmatization is dictionary-based and chooses the appropriate lemma based on context, stemming operates on single words without considering the context. For example, in the sentence:

"This is better"
The word "better" is transformed into the word "good" by a lemmatizer but is unchanged by stemming. Even though they can lead to less-accurate results, stemmers are easier to build and perform faster than lemmatizers. However, the latter is better if you're seeking more precise linguistic rules.

Stopword Removal
Removing stop words is an important step in NLP text processing. It involves filtering out high-frequency words that add little or no semantic value to a sentence, for example, *which, to, at, for, is,* etc.
You can even customize lists of stopwords to include words that you want to ignore.

Let's say you'd like to classify customer service tickets based on their topics. In this example: *"Hello, I'm having trouble logging in with my new password"*, it may be useful to remove stop words like *"hello"*, *"I"*, *"am"*, *"with"*, *"my"*, so you're left with the words that help you understand the topic of the ticket: *"trouble"*, *"logging in"*, *"new"*, *"password"*.

Semantic

Semantic analysis focuses on identifying the meaning of language. However, since language is polysemic and ambiguous, semantics is considered one of the most challenging areas in NLP.

Semantic tasks analyze the structure of sentences, word interactions, and related concepts, in an attempt to discover the meaning of words, as well as understand the topic of a text.

Some sub-tasks of semantic analysis include:

Word Sense Disambiguation
Depending on their context, words can have different meanings. Take the word *"book"*, for example:

- *You should read this **book**, it's a great novel!*
- *You should **book** the flights as soon as possible.*
- *You should close the **books** by the end of the year.*
- *You should do everything by the **book** to avoid potential complications.*

There are two main techniques that can be used for Word Sense Disambiguation (WSD): knowledge-based (or dictionary approach) and a supervised approach. The first one tries to infer

meaning by observing the dictionary definitions of ambiguous terms within a text; while the latter is based on machine learning algorithms that learn from examples (training data).

Relationship Extraction

This task consists of identifying semantic relationships between two or more entities in a text. Entities can be names, places, organizations, etc; and relationships can be established in a variety of ways. For example, in the phrase *"Susan lives in Los Angeles"*, a person (*Susan*) is related to a place (*Los Angeles*) by the semantic category *"lives in"*.

CHAPTER 3

MACHINE LEARNING NLP

3. MACHINE LEARNING NLP

There are two main technical approaches to Natural Language Processing that create different types of systems: one is based on linguistic rules and the other on machine learning methods. In this section, we'll examine the advantages and disadvantages of each one, and the possibility of combining both (hybrid approach).

3.1 Machine Learning Models

Machine Learning consists of algorithms that can learn to understand language based on previous observations. The system uses statistical methods to build its own 'knowledge bank', and is trained to make associations between a particular input and its corresponding output.

With machine learning, you can build a model to automatically classify opinions as positive, negative, or neutral. But first, you need to train your classifier by manually tagging text examples, until it's ready to make its own predictions for unseen data.

Transform the text examples into something a machine can understand (vectors), a process known as feature extractor or text vectorization. Once the texts have been transformed into vectors, they are fed to a machine learning algorithm together with their expected output (tags) to create a classification model. This model can then discern which features best represent the texts, and make predictions for unseen data:

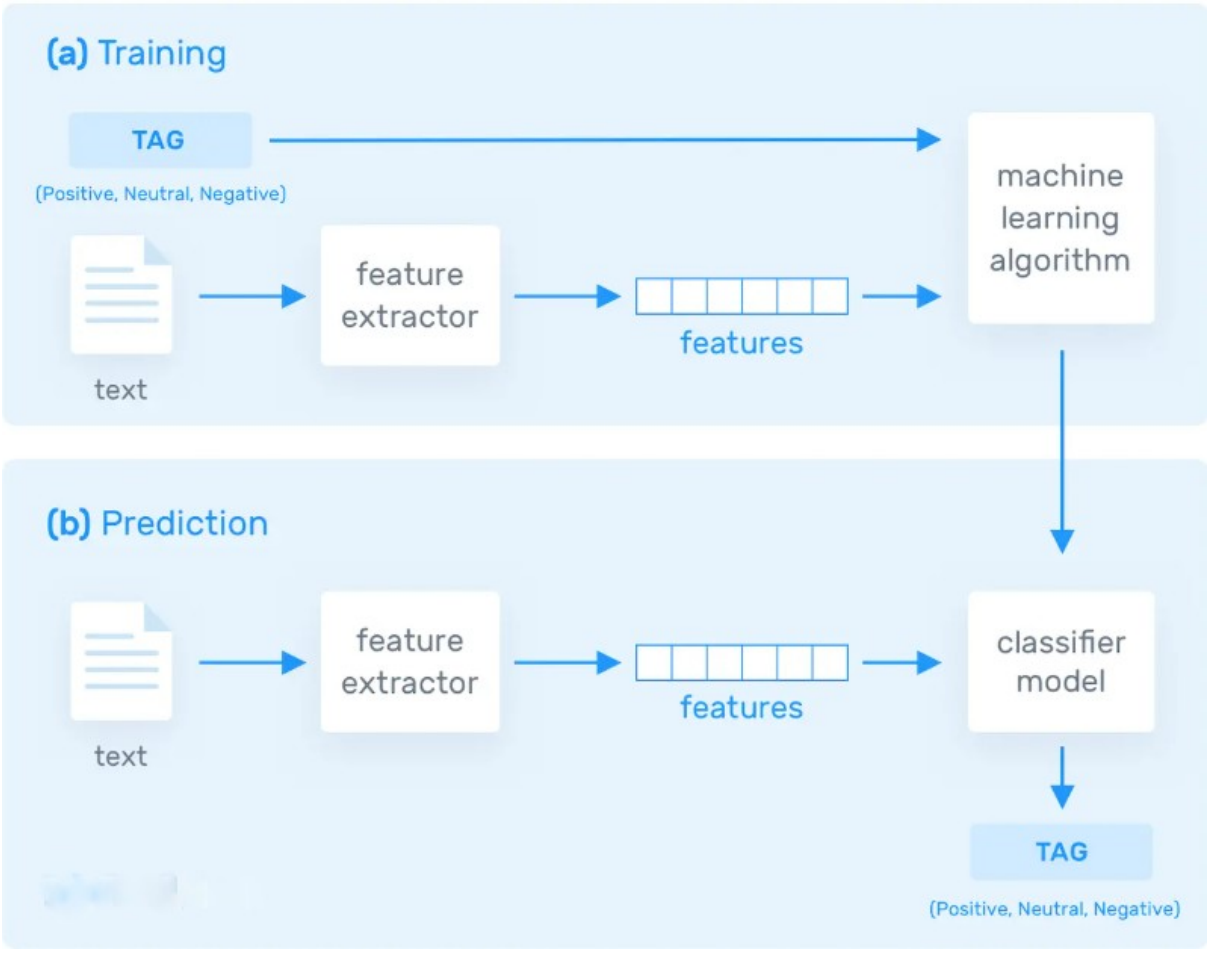


Fig 3.1: Training and Prediction

The biggest advantage of machine learning models is their ability to learn on their own, with no need to define manual rules. All you'll need is a good set of training data, with several examples for each of the tags you'd like to analyze.

Over time, machine learning models often deliver higher precision than rule-based systems, and the more training data you feed them, the more accurate they are.

However, you'll need training data that's relevant to the problem you want to solve in order to build an accurate machine learning model.

Hybrid Approach

A third approach involves combining both rule-based and machine learning systems. That way, you can benefit from the advantages of each of them, and gain higher accuracy in your results.

3.2 NLP Algorithms

Natural language processing algorithms are usually based on machine learning algorithms. Below are some of the most popular ones that you can use depending on the task you want to perform:

3.2.1 Text Classification Algorithms

Text classification is the process of organizing unstructured text into predefined categories (tags). Text classification tasks include sentiment analysis, intent detection, topic modeling, and language detection.
Some of the most popular algorithms for creating text classification models are:

- **Naive Bayes:** a collection of probabilistic algorithms that draw from the probability theory and Bayes' Theorem to predict the tag of a text. According to Bayes' Theorem, the probability of an event happening (A) can be calculated if a prior event (B) has happened.
  This model is called naive because it assumes that each variable (features or predictors) is independent, has no effect on the others, and each variable has an equal impact on the outcome. Naive Bayes algorithm is used for text classification, sentiment analysis, recommendation systems, and spam filters.

- **Support Vector Machines (SVM):** this is an algorithm mostly used to solve classification problems with high accuracy. Supervised classification models aim to predict the category of a piece of text based on a set of manually tagged training examples.
  In order to do that, SVM turns training examples into vectors and draws a hyperplane to differentiate two classes of vectors: those that belong to a certain tag and those that don't belong to that one tag. Based on which side of the boundary they land, the model will be able to assign one tag or another. SVM algorithms can be especially useful when you have a limited amount of data.

- **Deep Learning:** this set of machine learning algorithms are based on artificial neural networks. They are perfect for processing large volumes of data, but in turn, require a large training corpus. Deep learning algorithms are used to solve complex NLP problems.

3.2.2 Text Extraction Algorithms

Text extraction consists of extracting specific pieces of data from a text. You can use extraction models to pull out keywords, entities such as company names or locations, otherwise known as entity recognition, or to summarize text. Here are the most common algorithms for text extraction:

- **TF-IDF (term frequency-inverse document frequency):** this statistical approach determines how relevant a word is within a text in a collection of documents, and is often used to extract relevant keywords from text. The importance of a word increases based on the number of times it appears in a text (text frequency), but decreases based on the frequency it appears in the corpus of texts (inverse document frequency).
- **Regular Expressions (regex):** A regular expression is a sequence of characters that define a pattern. Regex checks if a string contains a determined search pattern, for example in text editors or search engines and is often used for extracting keywords and entities from text.
- **CRF (conditional random fields):** this machine learning approach learns patterns and extracts data by assigning a weight to a set of features in a sentence. This approach can create patterns that are richer and more complex than those patterns created with regex, enabling machines to determine better outcomes for more ambiguous expressions.
- **Rapid Automatic Keyword Extraction (RAKE):** this algorithm for keyword extraction uses a list of stopwords and phrase delimiters to identify relevant words or phrases within a text. Basically, it analyzes the frequency of a word and its co-occurrence with other words.

3.2.3 Topic Modeling Algorithms

Topic modeling is a method for clustering groups of words and similar expressions within a set of data. Unlike topic classification, topic modeling is an unsupervised method, which means that it infers patterns from data without needing to define categories or tag data beforehand.
The main algorithms used for topic modeling include:

- **Latent Semantic Analysis (LSA):** this method is based on the distributional hypothesis, and identifies words and expressions with similar meanings that occur in similar pieces of text. It is the most frequent method for topic modeling.
- **Latent Dirichlet Allocation (LDA):** this is a generative statistical model that assumes that documents contain various topics, and that each topic contains words with certain probabilities of occurrence.

CHAPTER -4
TEXTBLOB: SIMPLIFIED TEXT PROCESSING

4.TextBlob: Simplified Text Processing

TextBlob is a Python (2 and 3) library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more. TextBlob is a python library and offers a simple API to access its methods and perform basic NLP tasks. A good thing about TextBlob is that they are just like python strings.

4.1 NLP tasks using TextBlob

1.Tokenization

Tokenization refers to dividing text or a sentence into a sequence of tokens, which roughly correspond to “words”. This is one of the basic tasks of NLP. To do this using TextBlob, follow the two steps:

1. Create a **textblob** object and pass a string with it.
2. Call **functions** of textblob in order to do a specific task.

So, let’s quickly create a textblob object

```
from textblob import TextBlob

blob = TextBlob("Analytics Vidhya is a great platform to learn data science. \n It helps community through blogs, hackathons, discussions,etc.")
```

2 Noun Phrase Extraction

Since we extracted the words in the previous section, instead of that we can just extract out the noun phrases from the textblob. Noun Phrase extraction is particularly important when you want to analyze the “who” in a sentence. Lets see an example below.

```
blob = TextBlob("Analytics Vidhya is a great platform to learn data science.")

for np in blob.noun_phrases:

 print (np)
```

```
>> analytics vidhya

great platform

data science
```

As we can see that the results aren’t perfectly correct, but we should be aware that we are working with machines.

3 Part-of-speech Tagging

Part-of-speech tagging or grammatical tagging is a method to mark words present in a text on the basis of its definition and context. In simple words, it tells whether a word is a noun, or an adjective, or a verb, etc. This is just a complete version of noun phrase extraction, where we want to find all the the parts of speech in a sentence.

Let’s check the tags of our textblob.

```
for words, tag in blob.tags:

 print (words, tag)

>> Analytics NNS

Vidhya NNP

is VBZ

a DT

great JJ

platform NN

to TO

learn VB
```

```
data NNS

science NN
```

Here, NN represents a noun, DT represents as a determiner, etc. You can check the full list of tags from here to know more.

4 Words Inflection and Lemmatization

Inflection is a process of *word* formation in which characters are added to the base form of a *word* to express grammatical meanings. Word inflection in TextBlob is very simple, i.e., the words we tokenized from a textblob can be easily changed into singular or plural.

```
blob = TextBlob("Analytics Vidhya is a great platform to learn data science. \n It helps community through blogs, hackathons, discussions,etc.")

print (blob.sentences[1].words[1])

print (blob.sentences[1].words[1].singularize())

>> helps

help
```

TextBlob library also offers an in-build object known as *Word.* We just need to create a word object and then apply a function directly to it as shown below.

```
from textblob import Word

w = Word('Platform')

w.pluralize()

>>'Platforms'
```

We can also use the tags to inflect a particular type of words as shown below.

```
## using tags

for word,pos in blob.tags:

 if pos == 'NN':

 print (word.pluralize())

>> platforms

sciences
```

Words can be lemmatized using the *lemmatize* function.

```
## lemmatization

w = Word('running')

w.lemmatize("v") ## v here represents verb

>> 'run'
```

5 N-grams

A combination of multiple words together are called N-Grams. N grams (N > 1) are generally more informative as compared to words, and can be used as features for language modelling. N-grams can be easily accessed in TextBlob using the ***ngrams*** function, which returns a tuple of n successive words.

```
for ngram in blob.ngrams(2):

print (ngram)

>> ['Analytics', 'Vidhya']
```

```
['Vidhya', 'is']

['is', 'a']

['a', 'great']

['great', 'platform']

['platform', 'to']

['to', 'learn']

['learn', 'data']

['data', 'science']
```

6 Sentiment Analysis

Sentiment analysis is basically the process of determining the attitude or the emotion of the writer, i.e., whether it is positive or negative or neutral.
The sentiment function of textblob returns two properties, polarity, and subjectivity.
Polarity is float which lies in the range of [-1,1] where 1 means positive statement and -1 means a negative statement. Subjective sentences generally refer to personal opinion, emotion or judgment whereas objective refers to factual information. Subjectivity is also a float which lies in the range of [0,1].
Let’s check the sentiment of our blob.

```
print (blob)

blob.sentiment

>> Analytics Vidhya is a great platform to learn data science.

Sentiment(polarity=0.8, subjectivity=0.75)
```

We can see that polarity is **0.8**, which means that the statement is positive and **0.75** subjectivity refers that mostly it is a public opinion and not a factual information.

4.2 Features

1 Spelling Correction
2 Creating a short summary of a text
3 Translation and Language Detection
4. Text classification using TextBlob

4.3 Pros and Cons

Pros:
Since, it is built on the shoulders of NLTK and Pattern, therefore making it simple for beginners by providing an intuitive interface to NLTK.
It provides language translation and detection which is powered by Google Translate (not provided with Spacy).
Cons:
It is little slower in the comparison to spacy but faster than NLTK. (Spacy > TextBlob > NLTK)
It does not provide features like dependency parsing, word vectors etc. which is provided by spacy.

CHAPTER – 5
. NATURAL LANGUAGE PROCESSING EXAMPLES

5. NATURAL LANGUAGE PROCESSING EXAMPLES

NLP-powered systems, companies are able to automate tasks like ticket tagging, routing, and data entry, and gain fine-grained insights that can be used to make data-driven decisions.

Here are some examples of how NLP is used in business:

Quickly Sorting Customer Feedback

Text classification models are excellent for categorizing qualitative feedback, such as product reviews, social media conversations, and open-ended responses in online surveys.
Text classification models are excellent for categorizing qualitative feedback, such as responses to open-ended questions in online surveys.
Tagging each piece of feedback automatically with NLP tools enabled them to find out the most relevant topics mentioned by customers, along with how much they valued their product. As you can see in the graph below, most of the responses referred to “Product Features”, followed by “Product UX” and “Customer Support”.

Fig 5.1: Customer Feedback

Automating Processes in Customer Service

Other interesting applications of NLP revolve around customer service automation. This concept uses AI-based technology to eliminate or reduce routine manual tasks in customer support, saving agents valuable time, and making processes more efficient.

According to the Zendesk benchmark, a tech company receives +2600 support inquiries per month. Receiving large amounts of support tickets from different channels (email, social media, live chat, etc), means companies need to have a strategy in place to categorize each incoming ticket.
Text classification allows companies to automatically tag incoming customer support tickets according to their topic, language, sentiment, or urgency. Then, based on these tags, they can instantly route tickets to the most appropriate pool of agents.
Uber designed its own ticket routing workflow, which involves tagging tickets by *Country, Language, and Type* (this category includes the sub-tags *Driver-Partner, Questions about Payments, Lost Items, etc*), and following some prioritization rules, like sending requests from new customers (*New Driver-Partners*) to the top of the list.

Chatbots for Customer Success

A chatbot is a computer program that simulates human conversation. Chatbots use NLP to recognize the intent behind a sentence, identify relevant topics and keywords, verbs, and even emotions, and come up with the best response based on their interpretation of data.

As customers crave fast, personalized, and around-the-clock support experiences, chatbots have become the heroes of customer service strategies. Chatbots reduce customer waiting times by providing immediate responses and excel when handling routine queries (which often represent a high volume of customer support requests), allowing agents to focus on solving more complex issues. In fact, chatbots can solve up to 80% of routine customer support tickets.
Besides providing customer support, chatbots can be used to recommend products, offer discounts, and make reservations, among many other tasks. In order to do that, most chatbots follow a simple 'if/then' logic (they are programmed to identify intents and associate them with a certain action), or provide a selection of options to choose from.

Automatic Summarization

Automatic summarization consists of reducing a text and creating a concise new version that contains its most relevant information. It can be particularly useful to summarize large pieces of unstructured data, such as academic papers.

There are two different ways of using NLP for summarization: the first approach extracts the most important information within a text and uses it to create a summary *(extraction-based summarization)*; while the second applies deep learning techniques to paraphrase the text and produce sentences that are not present in the original source *(abstraction-based summarization)*.
Automatic summarization can be particularly useful for data entry, where relevant information is extracted from, let's say a product description, and automatically entered into a database.

Machine Translation

The possibility of translating text and speech to different languages has always been one of the main interests in the NLP field. From the first attempts to translate text from Russian to English in the 1950s to the state-of-the-art neural systems, machine translation (MT) has seen significant improvements but still presents challenges.

Google Translate, Microsoft Translator, and Facebook Translation App are a few of the leading platforms for generic machine translation. In August 2019, Facebook AI English-to-German machine translation model received first place in the contest held by the Conference of Machine Learning (WMT). The translations obtained by this model were defined by the organizers as "superhuman" and considered highly superior than the ones done by human experts.

Another interesting development in machine translation has to do with customizable machine translation systems, which are adapted to a specific domain and trained to understand the terminology associated with a particular field, such as medicine, law, and finance. Lingua Custodia, for example, is a machine translation tool dedicated to translating technical financial documents. Finally, one of the latest innovations in MT is adaptative machine translation, which consists of systems that can learn from corrections in real-time.

Natural Language Generation

Natural Language Generation (NLG) is a subfield of NLP designed to build computer systems or applications that can automatically produce all kinds of texts in natural language by using a semantic representation as input. Some of the applications of NLG are question answering and text summarization.

In 2019, artificial intelligence company Open AI released GPT-2, a text-generation system that represented a ground-breaking achievement in AI and has taken the NLG field to a whole new level. The system was trained with a massive dataset of 8 million web pages and it’s able to generate coherent and high-quality pieces of text (like news articles, stories, or poems), given minimum prompts.
The model performs better when provided with popular topics which have a high representation in the data (such as Brexit, for example), while it offers poorer results when prompted with highly niched or technical content. Still, its possibilities are only beginning to be explored.

CHAPTER -6

SOFTWARE-LIBRARIES

6 SOFTWARE-LIBRARIES

1.Scikit-learn

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language.[3] It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Scikit-learn is largely written in Python, and uses numpy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Cython to improve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. In such cases, extending these methods with Python may not be possible.

Scikit-learn integrates well with many other Python libraries, such as matplotlib and plotly for plotting, numpy for array vectorization, pandas data frames, scipy, and many more.

2. Pandas

Pandas is mainly used for data analysis. Pandas allows importing data from various file formats such as comma-separated values, JSON, SQL, Microsoft Excel [6]. Pandas allows various data manipulation operations such as merging [7], reshaping [8], selecting [9], as well as data cleaning, and data wrangling features.

When you want to use Pandas for data analysis, you'll usually use it in one of three different ways:

Convert a Python's list, dictionary or Numpy array to a Pandas data frame.

Open a local file using Pandas, usually a CSV file, but could also be a delimited text file (like TSV), Excel, etc.

3. NumPy

NumPy is a python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important. NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This behavior is called locality of reference in computer science.

This is the main reason why NumPy is faster than lists. Also, it is optimized to work with latest CPU architectures.

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

4. Matplotlib

Matplotlib is a data visualization library that is used for 2D plotting to produce publication-quality image plots and figures in a variety of formats.

The library helps to generate histograms, plots, error charts, scatter plots, bar charts with just a few lines of code.

It provides a MATLAB-like interface and is exceptionally user-friendly. It works by using standard GUI toolkits like GTK+, wxPython, Tkinter, or Qt to provide an object-oriented API that helps programmers to embed graphs and plots into their applications.

CHAPTER -7

ALGORITHMS

7.Algorithm

1)Support Vector Machine:

Step-1
Importing the libraries and Reading the dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('Restaurent_reviews.tsv', delimiter='\t', quoting=3)
```

Step -2
Importing re, nltk, stopwords and Porterstemmer

```
import re
import nltk
nltk.download('stopwords')
#nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
corpus = [] # list will contain all the refined reviews
```

```
[nltk_data] Downloading package stopwords to C:\Users\vaishnavi
[nltk_data]     m\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Step -3
Refining dataset, removing and adding stop words

```
for i in range(0,1000):
    review = df['Review'][i] # Collecting the reviews one by one
    review = re.sub('[^a-zA-z]', ' ', review) # replacing the punctuations with space
    review = review.lower() # converting all the characters into lower case
    review = review.split() # splitting word of the statement i.e converting a statement into list of words
    ps = PorterStemmer() # creating the object of the porter stemmer class
   # lemmatizer = WordNetLemmatizer()
    all_stopwords = stopwords.words('english') # collecting the english language stop words
    arr=['not','no',"shouldn't",'wasn','weren','won','didn']
    for i in arr:
        all_stopwords.remove(i)
    arr2=['order','waited',"bland",'still','place','feel','buffet']
    for i in arr2:
        all_stopwords.append(i)
    #all_stopwords.append('bar')
    #all_stopwords.append('20')
    #all_stopwords.append('disappoint')
    #review=[lemmatizer.lemmatize(word) for word in review if not word in set(all_stopwords)]
    review =[ps.stem(word) for word in review if not word in set(all_stopwords)]
    # coverting the list of words back to statement
    # for this we will use the join function
    review = ' '.join(review)
    corpus.append(review) # collecting the refined reviews
```

Step- 4

creating bag of words

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=1500,binary=True)
# using the max_feature parameter of countVectorizer to limit the number of columns in x

x = cv.fit_transform(corpus).toarray()
y=df.iloc[:,-1].values
```

Step-5

splitting x and y into training and test set

```
from sklearn.model_selection import train_test_split
x_tr,x_te,y_tr,y_te = train_test_split(x,y,test_size = 0.2, random_state = 0)
```

Step-6

creating and training the svm model

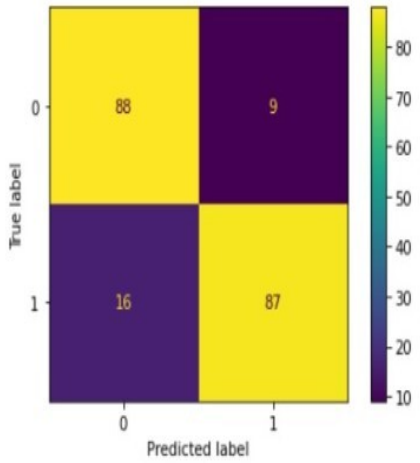
```
from sklearn.svm import SVC
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_tr,y_tr)
```

```
SVC(kernel='linear', random_state=0)
```

```
# predicting the output
y_pr = classifier.predict(x_te)
```

```
# plotting the confusion matrix and accuracy score
from sklearn.metrics import plot_confusion_matrix, accuracy_score
plot_confusion_matrix(estimator=classifier,X=x_te, y_true=y_te)
acc = accuracy_score(y_te,y_pr)
print(acc)
```

```
0.875
```



2)Principal Component Analysis (using SVM):

Step-1

Importing the libraries and Reading the dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('Restaurent_reviews.tsv', delimiter='\t', quoting=3)
```

Step -2

Importing re, nltk, stopwords and Porterstemmer

```
import re
import nltk
nltk.download('stopwords')
#nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
corpus = [] # list will contain all the refined reviews
```

```
[nltk_data] Downloading package stopwords to C:\Users\vaishnavi
[nltk_data]     m\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Step -3

Refining dataset, removing and adding stop words

```
for i in range(0,1000):
    review = df['Review'][i] # Collecting the reviews one by one
    review = re.sub('[^a-zA-z]', ' ', review) # replacing the punctuations with space
    review = review.lower() # converting all the characters into lower case
    review = review.split() # splitting word of the statement i.e converting a statement into list of words
    ps = PorterStemmer() # creating the object of the porter stemmer class
   # lemmatizer = WordNetLemmatizer()
    all_stopwords = stopwords.words('english') # collecting the english language stop words
    arr=['not','no',"shouldn't",'wasn','weren','won','didn']
    for i in arr:
        all_stopwords.remove(i)
    arr2=['order','waited',"bland",'still','place','feel','buffet']
    for i in arr2:
        all_stopwords.append(i)
    #all_stopwords.append('bar')
    #all_stopwords.append('20')
    #all_stopwords.append('disappoint')
    #review=[lemmatizer.lemmatize(word) for word in review if not word in set(all_stopwords)]
    review =[ps.stem(word) for word in review if not word in set(all_stopwords)]
    # coverting the list of words back to statement
    # for this we will use the join function
    review = ' '.join(review)
    corpus.append(review) # collecting the refined reviews
```

Step- 4

creating bag of words

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=1500,binary=True)
# using the max_feature parameter of countVectorizer to limit the number of columns in x

x = cv.fit_transform(corpus).toarray()
y=df.iloc[:,-1].values
```

Step-5

splitting x and y into training and test set

```
from sklearn.model_selection import train_test_split
x_tr,x_te,y_tr,y_te = train_test_split(x,y,test_size = 0.2, random_state = 0)
```

Step-6

Applying PCA

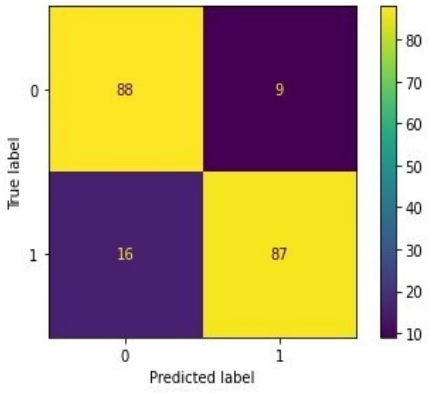
In [78]:
```
from sklearn.decomposition import PCA
pca=PCA(n_components=1000)
x=pca.fit_transform(x)
var=pca.explained_variance_ratio_
print(var)
```

```
[2.56332483e-02 2.20310711e-02 1.70950523e-02 1.60522870e-02
 1.54552267e-02 1.36259288e-02 1.16364821e-02 1.04754155e-02
 9.34025196e-03 8.70045758e-03 8.08019899e-03 7.90531585e-03
 7.74333094e-03 7.43429231e-03 7.25722780e-03 7.04778280e-03
 6.93147715e-03 6.67656439e-03 6.48934295e-03 6.35280891e-03
 6.27168371e-03 6.24506175e-03 5.92870124e-03 5.88147765e-03
 5.66814517e-03 5.64519534e-03 5.53972914e-03 5.51228464e-03
 5.40859357e-03 5.36766173e-03 5.25448568e-03 5.17870552e-03
 5.12682484e-03 5.04519672e-03 5.01994193e-03 4.87500327e-03
 4.83618073e-03 4.80747044e-03 4.72132041e-03 4.66435285e-03
 4.63730426e-03 4.56543999e-03 4.51173124e-03 4.45830962e-03
 4.42178806e-03 4.31225866e-03 4.24569359e-03 4.19144443e-03
 4.16143286e-03 4.14974506e-03 4.04868122e-03 3.97502929e-03
 3.95652130e-03 3.91113695e-03 3.87617369e-03 3.79659842e-03
 3.76865859e-03 3.72517398e-03 3.69497122e-03 3.65947175e-03
 3.63470383e-03 3.59508009e-03 3.57805866e-03 3.52456645e-03
 3.47884880e-03 3.42805326e-03 3.38722159e-03 3.35141225e-03
 3.32304848e-03 3.26217758e-03 3.23183427e-03 3.22126922e-03
 3.19035996e-03 3.18375272e-03 3.09998554e-03 3.09256808e-03
```

Step -7

:
```
# plotting the confusion matrix and accuracy score
from sklearn.metrics import plot_confusion_matrix, accuracy_score
plot_confusion_matrix(estimator=classifier,X=x_te, y_true=y_te)
acc = accuracy_score(y_te,y_pr)
print(acc)
```

```
0.875
```



–

3)K Nearest Neighbour (KNN):

Step-1
Importing the libraries and Reading the dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('Restaurent_reviews.tsv', delimiter='\t', quoting=3)
```

Step -2

Importing re, nltk, stopwords and Porterstemmer

```
import re
import nltk
nltk.download('stopwords')
#nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
corpus = [] # list will contain all the refined reviews
```

```
[nltk_data] Downloading package stopwords to C:\Users\vaishnavi
[nltk_data]     m\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Step -3

Refining dataset, removing and adding stop words

```
for i in range(0,1000):
    review = df['Review'][i] # Collecting the reviews one by one
    review = re.sub('[^a-zA-z]', ' ', review) # replacing the punctuations with space
    review = review.lower() # converting all the characters into lower case
    review = review.split() # splitting word of the statement i.e converting a statement into list of words
    ps = PorterStemmer() # creating the object of the porter stemmer class
   # lemmatizer = WordNetLemmatizer()
    all_stopwords = stopwords.words('english') # collecting the english language stop words
    arr=['not','no',"shouldn't",'wasn','weren','won','didn']
    for i in arr:
        all_stopwords.remove(i)
    arr2=['order','waited',"bland",'still','place','feel','buffet']
    for i in arr2:
        all_stopwords.append(i)
    #all_stopwords.append('bar')
    #all_stopwords.append('20')
    #all_stopwords.append('disappoint')
    #review=[lemmatizer.lemmatize(word) for word in review if not word in set(all_stopwords)]
    review =[ps.stem(word) for word in review if not word in set(all_stopwords)]
    # coverting the list of words back to statement
    # for this we will use the join function
    review = ' '.join(review)
    corpus.append(review) # collecting the refined reviews
```

Step- 4

creating bag of words

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=1500,binary=True)
# using the max_feature parameter of countVectorizer to limit the number of columns in x

x = cv.fit_transform(corpus).toarray()
y=df.iloc[:,-1].values
```

Step-5

splitting x and y into training and test set

```
from sklearn.model_selection import train_test_split
x_tr,x_te,y_tr,y_te = train_test_split(x,y,test_size = 0.2, random_state = 0)
```

Step-6

```
from sklearn.model_selection import train_test_split
x_tr,x_te,y_tr,y_te = train_test_split(x,y,test_size = 0.4, random_state = 0)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x = sc.fit_transform(x)
```

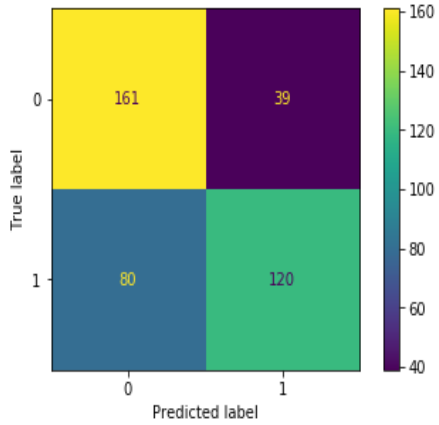
```
# creating and training the knn model
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=6,metric='minkowski',p=2)
classifier.fit(x_tr,y_tr)
```

```
KNeighborsClassifier(n_neighbors=6)
```

```
# predicting the output
y_pr = classifier.predict(x_te)
```

```
from sklearn.metrics import plot_confusion_matrix, accuracy_score, confusion_matrix
plot_confusion_matrix(estimator=classifier,X=x_te,y_true=y_te)
print(accuracy_score(y_te,y_pr))
```

```
0.7025
```



4 TEXTBLOB

Step -1

Importing the libraries, Reading the dataset and dividing total dataset into positive and negative reviews

```
from textblob import TextBlob
import nltk
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('Nlp.tsv', delimiter='\t', quoting=3)
print(df)
```

```
                                              Review  Liked
0                            Wow... Loved this place.      1
1                                  Crust is not good.      0
2           Not tasty and the texture was just nasty.      0
3    Stopped by during the late May bank holiday of...      1
4    The selection on the menu was great and so wer...      1
..                                                 ...    ...
995  I think food should have flavor and texture an...      0
996                            Appetite instantly gone.      0
997  Overall I was not impressed and would not go b...      0
998  The whole experience was underwhelming, and I ...      0
999  Then, as if I hadn't wasted enough of my life ...      0

[1000 rows x 2 columns]
```

```
pos=[]
neg=[]
for i in range(0,1000):
    if df['Liked'][i]==0:
        neg.append(df['Review'][i])
    else:
        pos.append(df["Review"][i])
```

Step -2

Splitting x and y into training and test set

```
from sklearn.model_selection import train_test_split
px_tr,px_te=train_test_split(pos,test_size=0.2)
nx_tr,nx_te=train_test_split(neg,test_size=0.2)
```

Step -3

Classifying reviews into positive and negative based on their polarities

```
pos_correct=0
pos_count=0
for i in px_tr:
  analysis=TextBlob(i)
  if analysis.sentiment.polarity >-0.1:
        pos_correct += 1
  pos_count +=1
neg_correct=0
neg_count=0
for i in nx_tr:
  analysis=TextBlob(i)
  if analysis.sentiment.polarity <=0.6:
      neg_correct += 1
  neg_count +=1
pos_acc=pos_correct/pos_count*100.0
neg_acc=neg_correct/neg_count*100.0
print("Train Positive accuracy = {}% via {} samples".format(pos_correct/pos_count*100.0, pos_count))
print("Train Negative accuracy = {}% via {} samples".format(neg_correct/neg_count*100.0, neg_count))
print("Train Total Accuracy= ",(((pos_correct/pos_count*100.0)+(neg_correct/neg_count*100.0))/2))
pos_correct=0
pos_count=0
for i in px_te:
  analysis=TextBlob(i)
  if analysis.sentiment.polarity >-0.1:
        pos_correct += 1
  pos_count +=1
neg_correct=0
neg_count=0
for i in nx_te:
  analysis=TextBlob(i)
  if analysis.sentiment.polarity <=0.6:
      neg_correct += 1
  neg_count +=1
pos_acc=pos_correct/pos_count*100.0
neg_acc=neg_correct/neg_count*100.0
print("Test Positive accuracy = {}% via {} samples".format(pos_correct/pos_count*100.0, pos_count))
print("Test Negative accuracy = {}% via {} samples".format(neg_correct/neg_count*100.0, neg_count))
print("Test Total Accuracy= ",(((pos_correct/pos_count*100.0)+(neg_correct/neg_count*100.0))/2))
```

Step -4

Output:

```
Train Positive accuracy = 94.75% via 400 samples
Train Negative accuracy = 97.5% via 400 samples
Train Total Accuracy=  96.125
Test Positive accuracy = 99.0% via 100 samples
Test Negative accuracy = 97.0% via 100 samples
Test Total Accuracy=  98.0
```

CHAPTER 8
COMPLETE CODE

8. COMPLETE CODE

1)Support Vector Machine:

```
#Importing the libraries
import numpy as np
import pandas as pd
#Reading the dataset
df = pd.read_csv('Restaurant_Reviews.tsv', delimiter='\t', quoting=3)

# Refining the data

import re

import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

from nltk.stem import WordNetLemmatizer

corpus = [] # list will contain all the refined reviews

for i in range(0,1000):

    review = df['Review'][i] # Collecting the reviews one by one

    review = re.sub('[^a-zA-z]', ' ', review) # replacing the punctuations with space

    review = review.lower() # converting all the characters into lower case

    review = review.split() # splitting word of the statement i.e converting a statement into list of
words

      ps = PorterStemmer() # creating the object of the porter stemmer class

    all_stopwords = stopwords.words('english') # collecting the english language stop words

    arr=['not','no',"shouldn't",'wasn','weren','won','didn']

    for i in arr:

        all_stopwords.remove(i)

    arr2=['order','waited',"bland",'still','place','feel','buffet']

         for i in arr2:

        all_stopwords.append(i)

    review =[ps.stem(word) for word in review if not word in set(all_stopwords)]

# converting the list of words back to statement

# for this we will use the join function
```

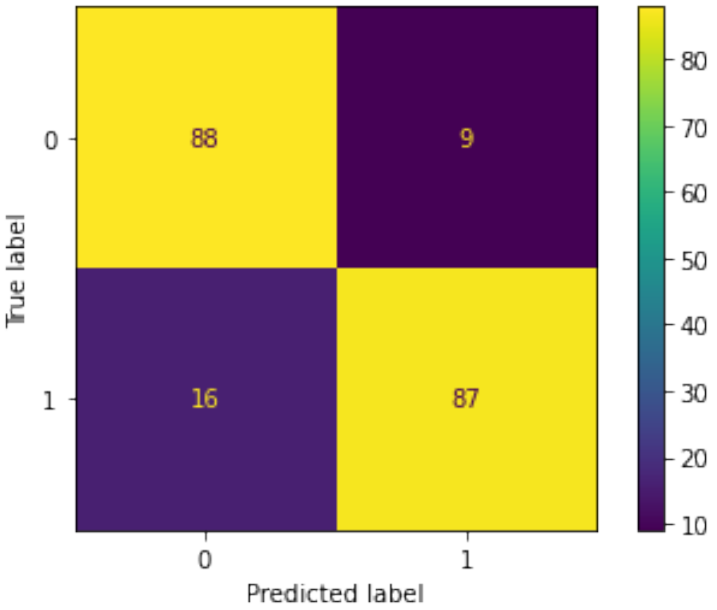
```
    review = ' '.join(review)
    corpus.append(review)# collecting the refined reviews

#Creating the Bag of Words
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=1600)
# using the max_feature parameter of countVectorizer to limit the number of columns in x
x = cv.fit_transform(corpus).toarray()
y = df.iloc[:,-1].values
#splitting x and y into training set
from sklearn.model_selection import train_test_split
x_tr,x_te,y_tr,y_te = train_test_split(x,y,test_size = 0.2, random_state = 0)
# Applying svm
from sklearn.svm import SVC
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_tr,y_tr)

y_pr = classifier.predict(x_te)
# plotting the confusion matrix and accuracy score
from sklearn.metrics import plot_confusion_matrix, accuracy_score
plot_confusion_matrix(estimator=classifier,X=x_te, y_true=y_te)
acc = accuracy_score(y_te,y_pr)
print('Accuracy is' ,acc*100)
```

output:

Accuracy is 87.5



2)Principal Component Analysis (using SVM):

```
#Importing the libraries
import numpy as np
import pandas as pd
#Reading the dataset
df = pd.read_csv('Restaurant_Reviews.tsv', delimiter='\t', quoting=3)

#  Refining the data

import re

import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

from nltk.stem import WordNetLemmatizer

corpus = [] # list will contain all the refined reviews

for i in range(0,1000):

    review = df['Review'][i] # Collecting the reviews one by one

    review = re.sub('[^a-zA-z]', ' ', review) # replacing the punctuations with space

    review = review.lower() # converting all the characters into lower case

    review = review.split() # splitting word of the statement i.e converting a statement into list        of words

    ps = PorterStemmer() # creating the object of the porter stemmer class

    all_stopwords = stopwords.words('english') # collecting the english language stop words

    arr=['not','no',"shouldn't",'wasn','weren','won','didn']
```

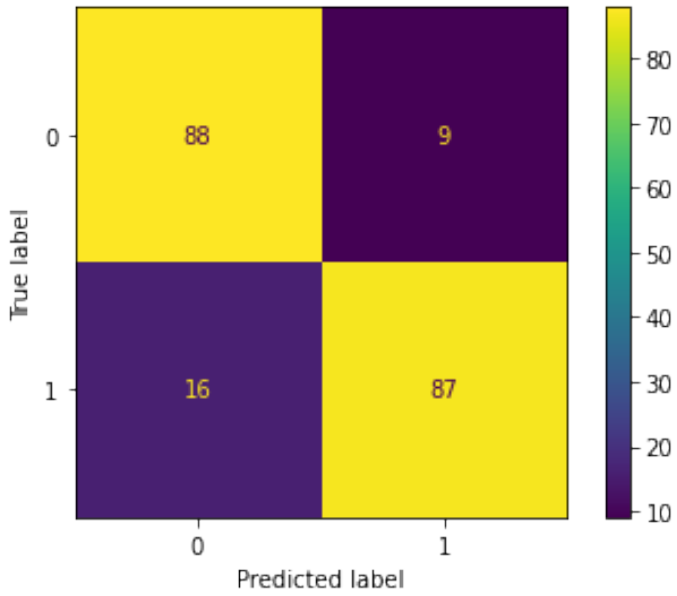
```
for i in arr:
    all_stopwords.remove(i)
arr2=['order','waited',"bland",'still','place','feel','buffet']
for i in arr2:
    all_stopwords.append(i)
review =[ps.stem(word) for word in review if not word in set(all_stopwords)]
# converting the list of words back to statement
# for this we will use the join function
review = ' '.join(review)
corpus.append(review)# collecting the refined reviews


#Creating the Bag of Words
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=1600)
# using the max_feature parameter of countVectorizer to limit the number of columns in x
x = cv.fit_transform(corpus).toarray()
y = df.iloc[:,-1].values
#splitting x and y into training set
from sklearn.model_selection import train_test_split
x_tr,x_te,y_tr,y_te = train_test_split(x,y,test_size = 0.2, random_state = 0)

#Applying PCA
from sklearn.decomposition import PCA
pca=PCA(n_components=800)
x=pca.fit_transform(x)
var=pca.explained_variance_ratio_
# Applying svm
from sklearn.svm import SVC
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_tr,y_tr)

y_pr = classifier.predict(x_te)
# plotting the confusion matrix and accuracy score
from sklearn.metrics import plot_confusion_matrix, accuracy_score
plot_confusion_matrix(estimator=classifier,X=x_te, y_true=y_te)
acc = accuracy_score(y_te,y_pr)
```

print(‘Accuracy is’ ,acc*100)



output:
Accuracy is 87.5

3)K Nearest Neighbour (KNN):

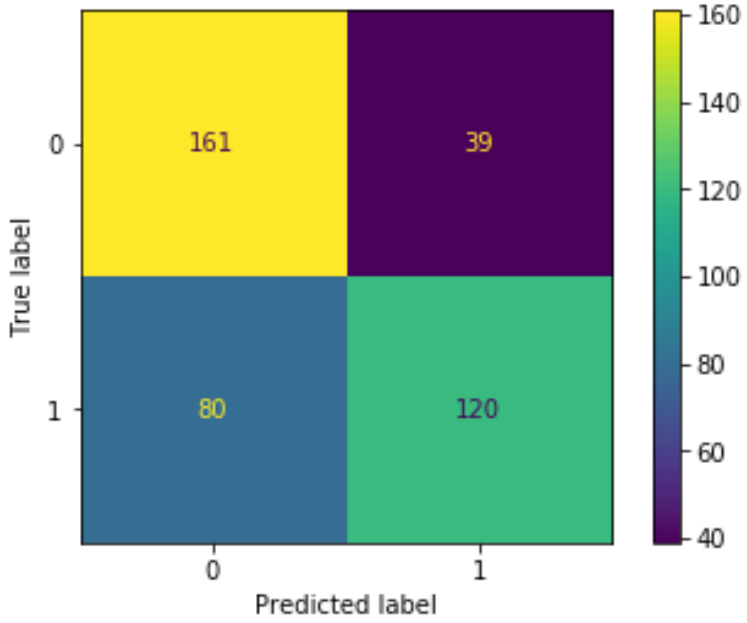
```
#Importing the libraries
import numpy as np
import pandas as pd
#Reading the dataset
df = pd.read_csv('Restaurant_Reviews.tsv', delimiter='\t', quoting=3)
```

```
# Refining the data
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
corpus = [] # list will contain all the refined reviews
for i in range(0,1000):
    review = df['Review'][i] # Collecting the reviews one by one
    review = re.sub('[^a-zA-z]', ' ', review) # replacing the punctuations with space
    review = review.lower() # converting all the characters into lower case
    review = review.split() # splitting word of the statement i.e converting a statement into list of words
    ps = PorterStemmer() # creating the object of the porter stemmer class
    all_stopwords = stopwords.words('english') # collecting the english language stop words
    arr=['not','no',"shouldn't",'wasn','weren','won','didn']
    for i in arr:
        all_stopwords.remove(i)
    arr2=['order','waited',"bland",'still','place','feel','buffet']
    for i in arr2:
        all_stopwords.append(i)
    review =[ps.stem(word) for word in review if not word in set(all_stopwords)]
      # converting the list of words back to statement
      # for this we will use the join function
```

```
    review = ' '.join(review)
    corpus.append(review)# collecting the refined reviews
#Creating the Bag of Words
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=1600)
# using the max_feature parameter of countVectorizer to limit the number of columns in x
x = cv.fit_transform(corpus).toarray()
y = df.iloc[:,-1].values
#splitting x and y into training set
from sklearn.model_selection import train_test_split
x_tr,x_te,y_tr,y_te = train_test_split(x,y,test_size = 0.2, random_state = 0)

#Applying KNN

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=6,metric='minkowski',p=2)
classifier.fit(x_tr,y_tr)
y_pr = classifier.predict(x_te)
from sklearn.metrics import plot_confusion_matrix, accuracy_score
plot_confusion_matrix(estimator=classifier,X=x_te, y_true=y_te)
acc = accuracy_score(y_te,y_pr)
print('Accuracy is ' ,acc*100)
output:
Accuracy is 70.25
```



4. Textblob

```
#Importing the libraries  and

from textblob import TextBlob
import nltk
import pandas as pd
df = pd.read_csv('Nlp.tsv', delimiter='\t', quoting=3)
print(df)

pos=[]
neg=[]
for i in range(0,1000):
    if df['Liked'][i]==0:
        neg.append(df['Review'][i])
    else:
        pos.append(df["Review"][i])

from sklearn.model_selection import train_test_split
px_tr,px_te=train_test_split(pos,test_size=0.2)
nx_tr,nx_te=train_test_split(neg,test_size=0.2)

pos_correct=0
pos_count=0
for i in px_tr:
  analysis=TextBlob(i)
  if analysis.sentiment.polarity >-0.1:
        pos_correct += 1
  pos_count +=1
neg_correct=0
neg_count=0
for i in nx_tr:
  analysis=TextBlob(i)
  if analysis.sentiment.polarity <=0.6:
     neg_correct += 1
  neg_count +=1
pos_acc=pos_correct/pos_count*100.0
neg_acc=neg_correct/neg_count*100.0
print("Train Positive accuracy = {}% via {} samples".format(pos_correct/pos_count*100.0,
pos_count))
print("Train Negative accuracy = {}% via {} samples".format(neg_correct/neg_count*100.0,
neg_count))
print("Train Total Accuracy=
",(((pos_correct/pos_count*100.0)+(neg_correct/neg_count*100.0))/2))
pos_correct=0
pos_count=0
for i in px_te:
```

```
  analysis=TextBlob(i)
  if analysis.sentiment.polarity >-0.1:
        pos_correct += 1
  pos_count +=1
neg_correct=0
neg_count=0
for i in nx_te:
  analysis=TextBlob(i)
  if analysis.sentiment.polarity <=0.6:
     neg_correct += 1
  neg_count +=1
pos_acc=pos_correct/pos_count*100.0
neg_acc=neg_correct/neg_count*100.0
print("Test Positive accuracy = {}% via {} samples".format(pos_correct/pos_count*100.0,
pos_count))
print("Test Negative accuracy = {}% via {} samples".format(neg_correct/neg_count*100.0,
neg_count))
print("Test Total Accuracy=
",(((pos_correct/pos_count*100.0)+(neg_correct/neg_count*100.0))/2))
```

output:

```
Train Positive accuracy = 96.0% via 400 samples
Train Negative accuracy = 96.75% via 400 samples
Train Total Accuracy=  96.375
Test Positive accuracy = 94.0% via 100 samples
Test Negative accuracy = 100.0% via 100 samples
Test Total Accuracy=  97.0
```

CHAPTER -9
CONCLUSION

Conclusion

In this project, we built a model that could reliably predict whether a given review about a restaurant is positive or negative. This model could predict almost all kinds of reviews into positive and negative data. This helps the people to choose the restaurant properly. While our model maybe inaccurate in some cases, we talked about how a dataset can contain inaccurate prediction of reviews. In our model, we used Textblob which gives us the polarity which helps us to find the sentimental value of a sentence. Using this project, we successfully created a NLP model that could give us an accuracy of 96.5%. We also studied NLP and machine learning approaches for sentimental analysis. In this project, we have also created 3 more models including SVM with an accuracy of 87.5%, KNN with an accuracy of 65.5% and SVM with PCA with an accuracy of 87.5%. In this project, we have learnt many different algorithms which improved our knowledge in the scope of NLTK and sentimental analysis. We have also learnt that NLP has many different applications in the fields like speech recognition, chatbots, market intelligence, text classification, character recognition, machine translation. We would like to conclude by saying we have enjoyed and simultaneously learned while making the project.