SENTIMENT ANALYSIS

A Project Report

In partial fulfilment of the requirements for the award of the degree

Bachelor of Engineering

In

Computer Science & Engineering

Under the guidance of Joyjit Guha Biswas

By

Shubhomay Kundu Poddar

Pitchika Harsh Rao

Pius Shaw

MCKV INSTITUTE OF ENGINEERING,

LILUAH

In association with

(ISO9001:2015)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V,
Kolkata, West Bengal 700091

Title of the Project: Sentiment Analysis
Project Members: Shubhomay Kundu Poddar, Pitchika Harsh Rao, Pius Shaw

Name of the guide: Mr. Joyjit Guha Biswas
Address: Ardent Computech Pvt. Ltd
(An ISO 9001:2015 Certified)
SDF Building, Module #132, Ground Floor,
Salt Lake City, GP Block, Sector V,
Kolkata,West Bengal, 700091

Project Version Control History:

Version	Primary Author	Description of Version	Date Completed
Final	Shubhomay Kundu Poddar, Pitchika Harsh Rao, Pius Shaw	Project Report	25th July, 2024

Signature of Team Member

Signature of Approver

Date:

Date:

For Office Use Only

Mr. Joyjit Guha Biswas

Project Proposal Evaluator

Declaration

We hereby declare that the project work being presented in the project proposal entitled “Sentiment Analysis” in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering at Ardent Computech PVT. LTD, Saltlake, Kolkata, West Bengal, is an authentic work carried out under the guidance of Mr. Joyjit Guha Biswas. The matter embodied in this project work has not been submitted elsewhere for the award of any degree of our knowledge and belief.

Date: 25/07/2024

Name of the Student:

i)Shubhomay Kundu Poddar
ii)Pitchika Harsh Rao
iii)Pius Shaw

Signature of the student:

Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V,
Kolkata, West Bengal 700091

Certificate

This is to certify that this proposal of minor project entitled "**Sentiment Analysis**" is a record of Bonafide work, carried out by Shubhomay Kundu Poddar, Pitchika Harsh Rao and Pius Shaw under my guidance at Ardent Computech PVT. LTD. In my opinion, the report in its present form is in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology and as per regulations of the Ardent®. To the best of our knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report.

Guide / Supervisor

Mr. Joyjit Guha Biswas

Project Engineer

Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V,
Kolkata, West Bengal 700091

Acknowledgement

Success of any project depends largely on the encouragement and guidelines of many others. We take this sincere opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project work.

We would like to show our greatest appreciation to Mr. Joyjit Guha Biswas, Project Engineer at Ardent, Kolkata. We always feel motivated and encouraged every time by his valuable advice and constant inspiration; without his encouragement and guidance this project would not have materialized.

Words are inadequate in offering our thanks to the other trainees, project assistants and other members at Ardent Computech Pvt. Ltd. for their encouragement and cooperation in carrying out this project work. The guidance and support received from all the members and who are contributing to this project, was vital for the success of this project.

Contents:

Overview:

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and has fewer syntactical constructions than other languages.

Python is interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to Perl and PHP.

Python is Interactive: You can actually sit at a Python prompt and interactwith the interpreter directly to write your programs.

Python is Object-Oriented: Python supports Object-Oriented style ortechnique of programming that encapsulates code within objects.

Python is a Beginner's Language: Python is a great language for the beginner-level programmers and supports the development of a wide range ofapplications from simple text processing to WWW browsers to games.

History of Python:

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small Talk, UNIX shell, and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Features of Python:

Easy-to-learn: Python has few Keywords, simple structure and clearly

defined syntax. This allows a student to pick up the language quickly.

Easy-to-Read: Python code is more clearly defined and visible to the eyes. Easy -to-Maintain: Python's source code is fairly easy-to-maintain.

A broad standard library: Python's bulk of the library is very portable and cross platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable: Python can run on the wide variety of hardware platforms and has the same interface on all platforms.

Extendable: You can add low level modules to the python interpreter. These modules enables programmers to add to or customize their tools to be more efficient.

Databases: Python provides interfaces to all major commercial databases.

GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Scalable: Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

- It support functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte code for building large applications.
- It provides very high level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collections.
- It can be easily integrated with C, C++, COM, Active X, CORBA and JAVA.

Environment Setup:

- 64-Bit OS
- Install Python 3
- Setup virtual environment
- Install Packages

Basic Syntax of Python Program:

Type the following text at the Python prompt and press the Enter –

>>> print "Hello, Python!"
If you are running new version of Python, then you would need to use print statement with parenthesis as in print ("Hello, Python!");.
However in Python version 2.4.3, this produces the following result –

Hello, Python!

Python Identifiers:

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9). Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive proming language.

Python Keywords:

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

For example:
And, exec, not Assert, finally, orBreak, for, pass Class, from, print ,continue, global, raisedef, if, return, del, import, tryelif, in, while else, is, with ,except, lambda, yield.

Lines & Indentation:

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example –

```
if True: print "True"
else:
print "False"
```

Command Line Arguments:

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h −

```
$ python-h
usage: python [option]...[-c cmd|-m mod | file |-][arg]...
```

Options and arguments (and corresponding environment variables):

-c cmd: program passed in as string(terminates option list)

-d : debug output from parser (also PYTHONDEBUG=x)

-E : ignore environment variables (such as PYTHONPATH)
-h : print this help message and exit [etc.]

Variable Types:

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Assigning Values to Variables:

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

```
counter=10          # An integer assignment
weight=10.60              # A floating point
name="Ardent"  # A string
```

Multiple Assignment:

Python allows you to assign a single value to several variables simultaneously. For example −

```
a = b = c = 1
a,b,c       =       1,2,"hello"
```

Standard Data Types:

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has five standard data types −

- String
- List
- Tuple
- Dictionary
- Number

Data Type Conversion:

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another.

Sr.No.	Function & Description
1	**int(x [,base])** Converts x to an integer. base specifies the base if x is a string
2	**long(x [,base])** Converts x to a long integer. base specifies the base if x is a string.
3	**float(x)** Converts x to a floating-point number.
4	**complex(real [,imag])** Creates a complex number.
5	**str(x)** Converts object x to a string representation.
6	**repr(x)** Converts object x to an expression string.
7	**eval(str)** Evaluates a string and returns an object.
8	**tuple(s)** Converts s to a tuple.
9	**list(s)** Converts s to a list.

Functions:

Defining a Function:

- def functionname(parameters):
  "function_docstring"
  function_suite
  return [expression]

Pass by reference vs Pass by value:

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example –

```
# Function definition is here

def changeme(mylist):
        "This changes a passed list into this function"
        mylist.append([1,2,3,4]);
        print"Values inside the function: ",mylist
        return

# Now you can call changeme function

mylist=[10,20,30];
changeme(mylist);
print"Values outside the function: ",mylist
```

Here, we are maintaining reference of the passed object and appending values in the same object. So, this would produce the following result −

```
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]
```

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope . For Example-

```
total=0;                # This is global variable.
```

```
# Function definition is here

def sum( arg1, arg2 ):

# Add both the parameters and return them."

total= arg1 + arg2;              # Here total is local variable.
print"Inside the function local total : ", total
return total;

# Now you can call sum functionsum(10,20);
print"Outside the function global total : ", total
```

When the above code is executed, it produces the following result −

```
Inside the function local total : 30
Outside the function global total : 0
```

Modules:

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference .

The Python code for a module named a name normally resides in a file named a name.py. Here's an example of a simple module, support.py

```
def print_func( par ):
        print"Hello : ",
        par return
```

The import Statement

You can use any Python source file as a module by executing an import statement in some other Python source file. The import has the following syntax –

```
import module1[, module2[,… moduleN]
```

Packages:

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub-subpackages, and so on.

Consider a file Pots.py available in Phone directory. This file has following line of source code −

```
def Pots():
        print "I'm Pots Phone"
```

Similar way, we have another two files having different functions with the same name as above –

Phone/Isdn.py file having function Isdn()
Phone/G3.py file having function G3()

Now, create one more file_init_.py in Phone directory −

Phone/_init_.py

To make all of your functions available when you've imported Phone, you need to put explicit import statements in _init_.py as follows −

```
from Pots import Pots
from Isdn import Isdn
from G3 import
```

Numpy:

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin.

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars.

Scipy:

Scientific computing tools for Python
SciPy refers to several related but distinct entities:

- The SciPy ecosystem, a collection of open source software for scientific computing in Python.
- The community of people who use and develop this stack.
- Several conferences dedicated to scientific computing in Python - SciPy, EuroSciPy, and SciPy.in.
- The SciPy library, one component of the SciPy stack, providing many numerical routines.

The SciPy ecosystem
Scientific computing in Python builds upon a small core of packages:

- Python, a general purpose programming language. It is interpreted and dynamically typed and is very well suited for interactive work and quick prototyping, while being powerful enough to write large applications in.
- NumPy, the fundamental package for numerical computation. It defines the numerical array and matrix types and basic operations on them.
- The SciPy library, a collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimization, statistics, and much more.
- Matplotlib, a mature and popular plotting package that provides publication-quality 2-D plotting, as well as rudimentary 3-D plotting.

On this base, the SciPy ecosystem includes general and specialised tools for data management and computation, productive experimentation, and high-performance computing. Below, we overview some key packages, though there are many more relevant packages.

Data and computation:

- pandas, providing high-performance, easy-to-use data structures.
- SymPy, for symbolic mathematics and computer algebra.
- NetworkX, is a collection of tools for analyzing complex networks.
- scikit-image is a collection of algorithms for image processing.
- scikit-learn is a collection of algorithms and tools for machine learning.
- h5py and PyTables can both access data stored in the HDF5 format.

Productivity and high-performance computing:

- IPython, a rich interactive interface, letting you quickly process data and test ideas.
- The Jupyter notebook provides IPython functionality and more in your web browser, allowing you to document your computation in an easily reproducible form.
- Cython extends Python syntax so that you can conveniently build C extensions, either to speed up critical code or to integrate with C/C++ libraries.
- Dask, Joblib or IPyParallel for distributed processing with a focus on numeric data.

Quality assurance:

- nose, a framework for testing Python code, being phased out in preference for pytest.
- numpydoc, a standard and library for documenting Scientific Python libraries.

Pandas:

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. "Panel data", an econometrics term for multidimensional, structured data sets.

Library features:

- Data Frame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and sub setting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.

- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation.

Python Speech Features:

This library provides common speech features for ASR including MFCCs and filterbank energies. If you are not sure what MFCCs are, and would like to know more have a look at this MFCC tutorial: http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/.

You will need numpy and scipy to run these files. The code for this project is available at https://github.com/jameslyons/python_speech_features .

Supported features:

- python_speech_features.mfcc() - Mel Frequency Cepstral Coefficients
- python_speech_features.fbank() - Filterbank Energies
- python_speech_features.logfbank() - Log Filterbank Energies
- python_speech_features.ssc() - Spectral Subband Centroids

To use MFCC features:

```
from python_speech_features import mfccfrom python_speech_features import
logfbankimport scipy.io.wavfile as wav

(rate,sig) = wav.read("file.wav")

mfcc_feat = mfcc(sig,rate)fbank_feat = logfbank(sig,rate)

print(fbank_feat[1:3,:])
```

OS: Miscellaneous operating system interfaces

This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see open(), if you want to manipulate paths, see the os.path module, and if you want to read all the lines in all the files on the command line see the fileinput module. For creating temporary files and directories see the tempfile module, and for high-level file and directory handling see the shutil module.

Notes on the availability of these functions:

The design of all built-in operating system dependent modules of Python issuch that as long as the same functionality is available, it uses the same interface; for example, the function os.stat(path) returns stat information about path in the same format (which happens to have originated with the POSIX interface).

Extensions peculiar to a particular operating system are also available through the os module, but using them is of course a threat to portability.

All functions accepting path or file names accept both bytes and string objects, and result in an object of the same type, if a path or file name is returned.

On VxWorks, os.fork, os.execv and os.spawn*p* are not supported.

Pickle: Python object serialization

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," 1 or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling".

Operator: Standard operators as functions

The operator module exports a set of efficient functions corresponding to the intrinsic operators of Python. For example, operator.add(x, y) is equivalent to the expression x+y. Many function names are those used for special methods, without the double underscores. For backward compatibility, many of these have a variant with the double underscores kept. The variants without the double underscores are preferred for clarity.
The functions fall into categories that perform object comparisons, logical operations, mathematical operations and sequence operations.

Tempfile: Generate temporary files and directories

This module creates temporary files and directories. It works on all supported platforms. TemporaryFile, NamedTemporaryFile, TemporaryDirectory, and SpooledTemporaryFile are high-level interfaces which provide automatic cleanup and can be used as context managers. mkstemp() and mkdtemp() are lower-level functions which require manual cleanup.

All the user-callable functions and constructors take additional arguments which allow direct control over the location and name of temporary files and directories. Files names used by this module include a string of random characters which allows those files to be securely created in shared temporary directories. To maintain backward compatibility, the argument order is somewhat odd; it is recommended to use keyword arguments for clarity.

Scikit-learn: A Powerful Tool for Machine Learning

Scikit-learn is a free and open-source Python library that provides a wide range of supervised and unsupervised machine learning algorithms. Built on top of NumPy, SciPy, and Matplotlib, it offers a consistent and efficient interface for data preprocessing, model training, evaluation, and prediction.

Key Features:

Comprehensive Algorithms: Covers a vast array of algorithms including:

- Classification (SVM, random forests, logistic regression, Naive Bayes)
- Regression (linear regression, ridge regression, lasso, decision trees)
- Clustering (k-means, DBSCAN)
- Model Selection (cross-validation, grid search)
- Preprocessing (feature scaling, normalization, encoding)

User-Friendly Interface: Provides a consistent API for various algorithms, making it easy to experiment with different models.

Efficiency: Leverages optimized implementations in C and Cython for performance.

Extensibility: Allows custom estimators to be created for specific problems.

Integration: Works seamlessly with other Python data science libraries like NumPy, Pandas, and Matplotlib.

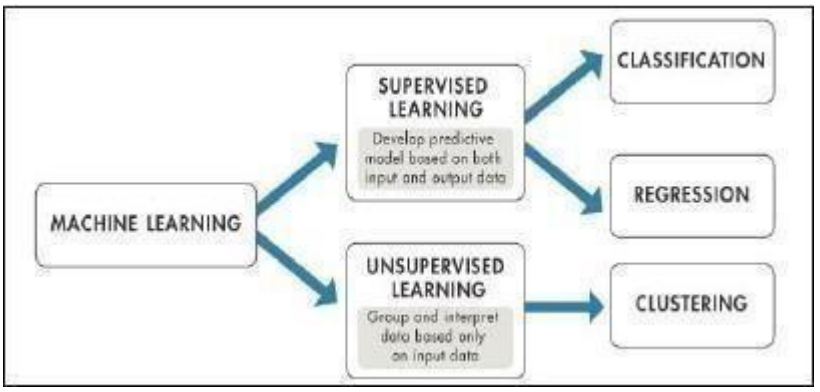
NLTK: A Toolkit for Natural Language Processing

NLTK (Natural Language Toolkit) is a powerful Python library specifically designed for working with human language data. It provides a comprehensive suite of tools for various natural language processing (NLP) tasks, making it a popular choice for both beginners and experienced practitioners.

Key Features of NLTK:

- **Tokenization:** Breaking text into words, sentences, or other meaningful units.
- **Stemming and Lemmatization:** Reducing words to their root form.
- **Part-of-Speech (POS) Tagging:** Identifying the grammatical role of words in a sentence.
- **Parsing:** Analyzing the grammatical structure of sentences.
- **Sentiment Analysis:** Determining the sentiment expressed in a text (positive, negative, neutral).
- **Corpus and Lexicon:** Access to a variety of text corpora and lexical resources.

Introduction to Machine Learnning:



Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.
Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data.

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.

Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:-

Supervised Learning:

Supervised learning is the machine learning task of inferring a function from labeled training data.[1] The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value.

A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. An optimal

scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

Unsupervised Learning:

Unsupervised learning is the machine learning task of inferring a function to describe hidden structure from "unlabelled" data (a classification or categorization is not included in the observations). Since the examples given tothe learner are unlabelled, there is no evaluation of the accuracy of the structurethat is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from supervised learning and reinforcement learning.

A central case of unsupervised learning is the problem of density estimation in statistics, though unsupervised learning encompasses many other problems (andsolutions) involving summarizing and explaining key features of the data.

Classification Algorithms Analysis:

Classification is a supervised machine learning technique that involves predicting the category or class to which new data points belong based on labeled training data. It's a fundamental task with applications across various domains, from spam filtering to medical diagnosis.

Popular Classification Algorithms:

- **Logistic Regression:** Predicts the probability of a data point belonging to a particular class.
- **Decision Trees:** Creates a tree-like model of decisions and their possible consequences.
- **Random Forest:** An ensemble method that combines multiple decision trees for improved accuracy.
- **Support Vector Machines (SVM):** Finds the optimal hyperplane to separate data points into different classes.
- **Naive Bayes:** Based on Bayes' theorem, assumes independence between features.
- **K-Nearest Neighbors (KNN):** Classifies new data points based on the majority class among its k nearest neighbors.

In our project we have implemented three classification models and their respective algorithms are given below.

Multinomial Naive Bayes algorithm

Step-by-Step Explanation:

1.Preprocessing:

1. Convert the text data into a format suitable for the algorithm. This usually involves tokenizing the text (breaking it down into words or tokens) and then converting these tokens into numerical features (e.g., using term frequency or TF-IDF).

2.Calculate Priors:

1. Calculate the prior probability for each class. The prior probability P(c)P(c)P(c) for a class ccc is given by the fraction of documents belonging to that class.

$$P(c) = \frac{\text{number of documents in class } c}{\text{total number of documents}}$$

3.Calculate Likelihoods:

1. Calculate the likelihood of each feature (token) given a class. This is the conditional probability P(xi ∣ c)P(x_i|c)P(xi ∣ c), where xix_ixi is the i-th token in the document. This can be done using the following formula:

$$P(x_i|c) = \frac{N_{x_i,c} + 1}{N_c + V}$$

Where:

- $N_{x_i,c}$ is the number of times token x_i appears in documents of class c.
- N_c is the total number of tokens in documents of class c.
- V is the total number of unique tokens in the training set.
- The "+1" in the numerator is used for Laplace smoothing to handle tokens that may not appear in the training data for a particular class.

4.Classification:

1. For a new document, calculate the posterior probability for each class. The posterior probability P(c ∣ d)P(c|d)P(c ∣ d) for a document belonging to class is given by:

$$P(c|d) \propto P(c) \prod_{i=1}^{n} P(x_i|c)$$

Where xi are the tokens in the document d and n is the number of tokens in the document. The class with the highest posterior probability is chosen as the predicted class for the document.

Logistic Regression algorithm

Step-by-Step Explanation

1.Initialization:

1. Initialize the weights (coefficients) and bias term. Typically, these are initialized to small random values or zeros.

2.Model Hypothesis:

1. The hypothesis for logistic regression is given by the logistic (sigmoid) function applied to the linear combination of the input features:

$$h_\theta(x) = \sigma(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

Where:

- θ is the vector of weights (including the bias term).
- x is the vector of input features.
- $\sigma(z)$ is the sigmoid function.

3.Cost Function:

1. The cost function used in logistic regression is the log-loss (or binary cross-entropy loss):

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

Where:

- m is the number of training examples.
- $y^{(i)}$ is the actual label for the i-th training example.
- $h_\theta(x^{(i)})$ is the predicted probability for the i-th training example.

4.Gradient Descent:

1. Update the weights and bias to minimize the cost function using gradient descent:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

Where:

- α is the learning rate.
- $\frac{\partial J(\theta)}{\partial \theta_j}$ is the gradient of the cost function with respect to the j-th weight.

The gradient of the cost function is given by:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

5.Prediction:

1. For a new input xxx, the prediction is made by computing the probability using the sigmoid function and then thresholding at 0.5:

$$\hat{y} = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5 \\ 0 & \text{if } h_\theta(x) < 0.5 \end{cases}$$

Support Vector Classifier algorithm

Step-by-Step Explanation

1.Define the Hyperplane:

1. For a binary classification problem, the goal is to find a hyperplane that best separates the two classes. In a 2-dimensional space, a hyperplane is simply a line, while in higher dimensions, it is a flat affine subspace.

2.Maximize the Margin:

1. The margin is defined as the distance between the hyperplane and the closest data points from each class, which are known as support vectors. SVM aims to maximize this margin.

3.Formulate the Optimization Problem:

1. The optimization problem can be formulated as follows:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

Subject to:

$$y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 \quad \text{for all } i$$

Where:

- $\mathbf{w}$ is the weight vector.
- b is the bias term.
- $y^{(i)}$ is the label of the i-th training example.
- $\mathbf{x}^{(i)}$ is the feature vector of the i-th training example.

4.Use the Kernel Trick (if necessary):

1. For non-linearly separable data, SVM can use kernel functions to project the data into a higher-dimensional space where a linear hyperplane can separate the classes. Common kernels include linear, polynomial, radial basis function (RBF), and sigmoid.

5.Solve the Optimization Problem:

1. This can be done using optimization techniques such as quadratic programming. The dual form of the optimization problem is often used for this purpose.

6.Make Predictions:

1. For a new data point x\mathbf{x}x, the prediction is made by computing the sign of the decision function:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

If f(x)≥0f(\mathbf{x}) \geq 0f(x)≥0, the data point is classified as one class, otherwise, it is classified as the other class.

Sentiment Analysis

Sentiment analysis is a challenging and essential task, impacting various sectors such as marketing, customer service, and social media monitoring. In this project, we aimed to create a sentiment analysis model using the above discussed classification algorithm approach and use of NLP.

Sentiment analysis is a complex task that involves numerous variables and non-linear interactions. While the algorithms can be a useful tool for certain simple sentiment analysis scenarios, it may not be sufficient for accurate predictions over more extensive datasets or complex linguistic patterns.

Here are the steps to create classification models for sentiment analysis:

1. Data Collection: Gather sentiment data from reliable sources, ensuring a diverse range of sentiments and topics.

2. Data Preprocessing: Clean the data, handle missing values, and ensure that all features are in a suitable format for analysis. This includes tokenization, removing stop words, and possibly stemming or lemmatization.

3. Feature Selection: Choose relevant features that might have a significant impact on sentiment classification. Perform exploratory data analysis to identify important features, which might include word frequencies, n-grams, or other text-based features.

4. Train-Test Split: Split the data into a training set and a testing set. The training set will be used to train the model, while the testing set will be used to evaluate its performance.

5. Model Training: Apply the Classifier algorithms to the training data. The model will learn the decision boundaries that separate different sentiment classes.

6. Model Evaluation: Use the testing set to evaluate the performance of the model. Common metrics for classification tasks include accuracy, precision, recall, and F1-score.

7. Model Optimization: Depending on the results, you might need to adjust the feature selection, consider additional features, or try other classification models or neural networks if it doesn't perform well. Techniques such as hyperparameter tuning can also be used to improve model performance

Actual codes for Sentiment Analysis:
Imports:

```
In [1]: import pandas as pd
        import numpy as np
        import re
        from sklearn.model_selection import train_test_split
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.svm import SVC
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import classification_report, accuracy_score,confusion_matrix
        from nltk.corpus import stopwords
        from nltk.stem import WordNetLemmatizer
        from nltk.tokenize import word_tokenize
        import nltk
```

Load the dataset:

```
In [3]: # Load the data
        data = pd.read_csv(r"chat_dataset.csv")
        data.head()
```

Out[3]:

	message	sentiment
0	I really enjoyed the movie	positive
1	The food was terrible	negative
2	I'm not sure how I feel about this	neutral
3	The service was excellent	positive
4	I had a bad experience	negative

Check for null values and count numbers of positive, negative and neutral sentiments:

```
In [4]: #Checking for null values and count numbers of positive, negative and neutral sentiments
        print(data.isnull().sum())
        data['sentiment'].value_counts()

        message      0
        sentiment    0
        dtype: int64

Out[4]: neutral     259
        positive    178
        negative    147
        Name: sentiment, dtype: int64
```

Data Preprocessing

In [5]:
```
# Data Preprocessing
#i)Cleaning text
def clean_text(text):
    """
    Cleans the input text by removing noise and normalizing the text.
    """
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special characters and numbers
    text = text.lower()  # Convert to lowercase
    return text
```

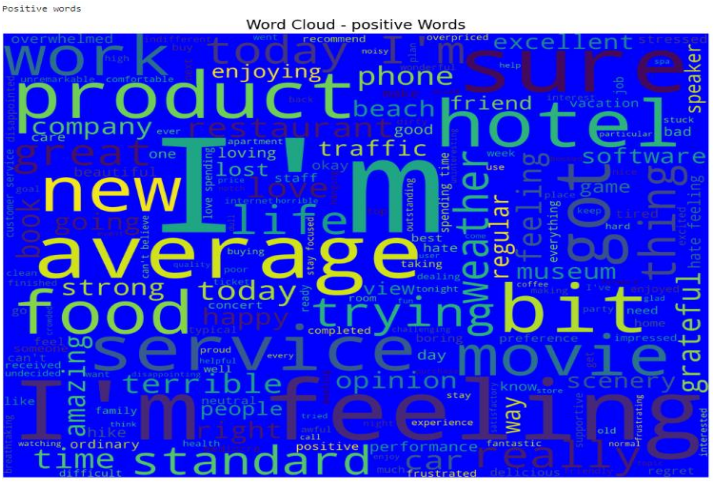
In [6]:
```
#ii)Preprocessing text
def preprocess_text(text):
    """
    Tokenizes the text, removes stopwords, and lemmatizes the words.
    """
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return ' '.join(tokens)
```

In [7]:
```
# Apply cleaning and preprocessing
data['cleaned_text'] = data['message'].apply(clean_text)
data['processed_text'] = data['cleaned_text'].apply(preprocess_text)
```

Wordcloud generation:

In [8]:
```
#Wordcloud generation
import matplotlib.pyplot as plt
from wordcloud import WordCloud  # Import for word cloud generation
pos = data[data['processed_text'] == 'positive']
pos = data['message']
neu = data[data['processed_text'] == 'neutral']
neu = data['message']
neg = data[data['processed_text'] == 'negative']
neg = data['message']
def wordcloud_draw(data, color='black'):
    """
    Generates and displays a word cloud visualization of text data.
    """
    words = ' '.join(data)  # Combine elements in data with spaces

    stop_words = set(stopwords.words('english'))
    # Create the word cloud object with desired parameters
    wordcloud = WordCloud(stopwords=stop_words,
                          background_color=color,
                          width=2500,
                          height=2000).generate(words)

    # Generate the plot
    plt.figure(figsize=(13, 13))  # Set figure size
    plt.imshow(wordcloud)
    plt.axis('off')  # Hide axis labels
    plt.title(f"Word Cloud - {'positive' if color == 'blue' else('negative' if color=='red' else 'neutral')} Words", fontsize=1
    plt.show()


# Assuming pos,neg and neu are available (lists containing positive,negative and neutral text data)
print("Positive words")
wordcloud_draw(pos, 'blue')

print("Negative words")
wordcloud_draw(neg, 'red')

print("Neutral words")
wordcloud_draw(neu, 'white')
```

Positive Words:



Negative Words:

Neutral Words:

Vectorizing The Data:

```
In [9]: # TF-IDF representation
        tfidf_vectorizer = TfidfVectorizer()
        X_tfidf = tfidf_vectorizer.fit_transform(data['processed_text'])
```

Train Test Split:

```
In [11]: # Split the data into training and testing sets
         X_train_tfidf, X_test_tfidf,y_train,y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)
```

Training the models:

1. Naive Bayes Classifier:

```
In [12]: # Model Training
         # 1. Naive Bayes
         nb_model = MultinomialNB()
         nb_model.fit(X_train_tfidf, y_train)
         y_pred_nb = nb_model.predict(X_test_tfidf)
```

```
In [13]: print("Naive Bayes Classifier:")
         print(classification_report(y_test, y_pred_nb))
         print(f"Accuracy: {accuracy_score(y_test, y_pred_nb)}\n")
```

```
Naive Bayes Classifier:
              precision    recall  f1-score   support

          -1       1.00      0.33      0.50        30
           0       0.67      0.98      0.80        54
           1       0.86      0.73      0.79        33

    accuracy                           0.74       117
   macro avg       0.84      0.68      0.69       117
weighted avg       0.81      0.74      0.72       117

Accuracy: 0.7435897435897436
```

2. Logistic Regression:

In [16]:
```
# 3. Logistic Regression
lr_model = LogisticRegression(max_iter=500)
lr_model.fit(X_train_tfidf, y_train)
y_pred_lr = lr_model.predict(X_test_tfidf)
```

In [17]:
```
print("Logistic Regression:")
print(classification_report(y_test, y_pred_lr))
print(f"Accuracy: {accuracy_score(y_test, y_pred_lr)}\n")
```

```
Logistic Regression:
              precision    recall  f1-score   support

          -1       1.00      0.53      0.70        30
           0       0.74      0.96      0.84        54
           1       0.84      0.79      0.81        33

    accuracy                           0.80       117
   macro avg       0.86      0.76      0.78       117
weighted avg       0.84      0.80      0.79       117

Accuracy: 0.8034188034188035
```

3. Support Vector Machine:

In [14]:
```
# 2. Support Vector Machine (SVM)
svm_model = SVC(kernel='linear')
svm_model.fit(X_train_tfidf, y_train)
y_pred_svm = svm_model.predict(X_test_tfidf)
```

In [15]:
```
print("Support Vector Machine:")
print(classification_report(y_test, y_pred_svm))
print(f"Accuracy: {accuracy_score(y_test, y_pred_svm)}\n")
```

```
Support Vector Machine:
              precision    recall  f1-score   support

          -1       0.86      0.63      0.73        30
           0       0.84      0.91      0.88        54
           1       0.78      0.88      0.83        33

    accuracy                           0.83       117
   macro avg       0.83      0.81      0.81       117
weighted avg       0.83      0.83      0.82       117

Accuracy: 0.8290598290598291
```

We go with Support Vector Classifier model since it is giving best accuracy out of all classification based models trained i.e. Logistic Regression and Multinomial Naive Bayes.

Hyper-parameter tunning with Support Vector Classifier model:

In [20]:
```
# SVM with Grid Search for hyperparameter tuning
from sklearn.model_selection import GridSearchCV
param_grid={'C':[0.9,1,1.2,1.1],'kernel' : ['linear', 'poly', 'rbf', 'sigmoid'],'degree':[2,3,4]}
grid_search = GridSearchCV(SVC(max_iter = 1000), param_grid, cv=5,n_jobs=-1)
grid_search.fit(X_train_tfidf, y_train)
```

Out[20]:
```
GridSearchCV(cv=5, estimator=SVC(max_iter=1000), n_jobs=-1,
             param_grid={'C': [0.9, 1, 1.2, 1.1], 'degree': [2, 3, 4],
                         'kernel': ['linear', 'poly', 'rbf', 'sigmoid']})
```

In [21]:
```
# Best model and evaluation
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_tfidf)
print(f"Best parameters: {grid_search.best_params_}")
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print(classification_report(y_test, y_pred))
```

```
Best parameters: {'C': 1, 'degree': 2, 'kernel': 'sigmoid'}
Accuracy: 0.8461538461538461
              precision    recall  f1-score   support

          -1       0.91      0.67      0.77        30
           0       0.85      0.93      0.88        54
           1       0.81      0.88      0.84        33

    accuracy                           0.85       117
   macro avg       0.85      0.82      0.83       117
weighted avg       0.85      0.85      0.84       117
```

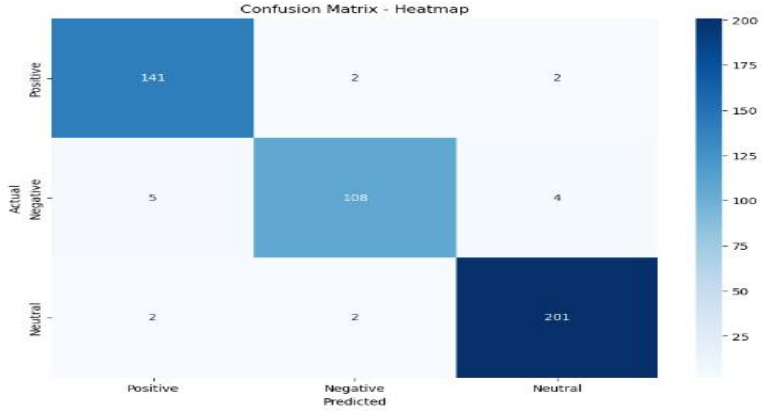
In [22]:
```
print(f"Accuracy: {accuracy_score(y_test, y_pred)}\n")
```

```
Accuracy: 0.8461538461538461
```

Printing confusion matrix: on train data

In [23]:
```
#Printing confusion matrix of the Support Vector Classifier model trained and tested on train data
y_pred = best_model.predict(X_train_tfidf)
# Print confusion matrix
conf_matrix = confusion_matrix(y_true=y_train, y_pred=y_pred, labels=[1,-1,0])
print("Confusion Matrix:")
print(conf_matrix)
import seaborn as sns
plt.figure(figsize=(10,7))
sns.heatmap(conf_matrix, annot=True, yticklabels=['Positive','Negative','Neutral'],xticklabels=['Positive','Negative','Neut
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Heatmap')
plt.show()
```

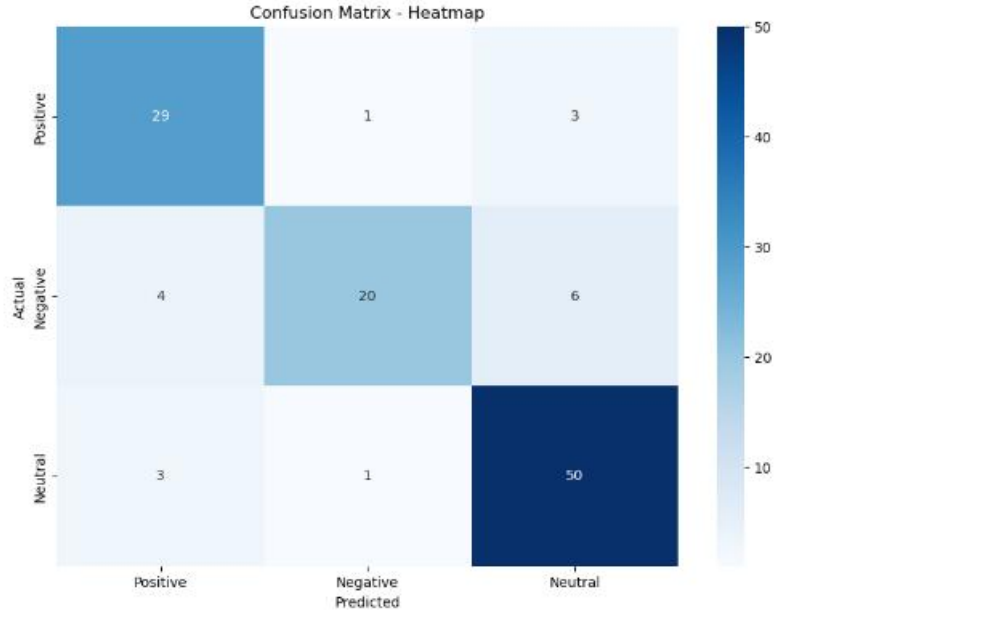
```
Confusion Matrix:
[[141   2   2]
 [  5 108   4]
 [  2   2 201]]
```



Printing confusion matrix: on test data

In [24]:
```
#Printing confusion matrix of the Support Vector Classifier model trained and tested on test data
y_pred = best_model.predict(X_test_tfidf)
# Print confusion matrix
conf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred, labels=[1,-1,0])
print("Confusion Matrix:")
print(conf_matrix)
import seaborn as sns
plt.figure(figsize=(10,7))
sns.heatmap(conf_matrix, annot=True, yticklabels=['Positive','Negative','Neutral'],xticklabels=['Positive','Negative','Neutral']
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Heatmap')
plt.show()
```

```
Confusion Matrix:
[[29  1  3]
 [ 4 20  6]
 [ 3  1 50]]
```



Conclusion:

We have collected raw data of chats from various online sources. After preprocessing this data, we transformed it into vectors suitable for machine learning algorithms. Through research and experimentation, we chose to use the Support Vector Machine (SVM) algorithm for this problem since it is giving good accuracy. Step by step, we developed the necessary functions with the help of online resources and our own insights. Despite encountering several errors, we persisted and successfully built a program that delivers the desired output. By testing with various splits of training and testing sets, we achieved a final accuracy of 84.6%. Given that this level of accuracy is satisfactory for sentiment analysis, we decided to save this model as our final classifier. This project served as a valuable introduction to machine learning in sentiment analysis and highlighted the importance of considering advanced models in practical applications.
Chat sentiment analysis helps in customer service and support, healthcare and mental health monitoring, product development and innovation and market research and competitive analysis by consideration of feedbacks

Future Scope:

For future iterations of this project, we plan to:

- Explore advanced sentiment analysis models: Investigate the use of deep learning models, such as recurrent neural networks (RNNs) or transformers, which can capture the nuances in textual data more effectively.
- Ensemble methods: Employ techniques like Random Forest or Gradient Boosting to combine predictions from multiple models and enhance accuracy.
- Expand the scope: Extend the project to predict other aspects of emotions, such as intensity or context, and apply it to different domains like customer feedback analysis or mental health monitoring.
- Implement real-time sentiment analysis: Develop a system capable of analyzing emotions in real-time, which can be useful in applications like chatbots, live feedback systems, or social media monitoring.

This project lays a solid foundation for further exploration and development in the field of sentiment analysis, with numerous opportunities for improvement and expansion.

THANK YOU