**Sentiment Analysis**

*BSES Yamuna Power Limited*

1. **INTRODUCTION**

By analyzing Twitter feedback, we can classify sentiments into positive, neutral, and negative categories, enabling companies to understand customer emotions and respond proactively to their needs. This documentation outlines the methodology, implementation, and findings of the sentiment analysis project conducted on Twitter and Facebook feedback received by the company.

**REQUIREMENTS:**

Sentiment analysis is an important part of the process that ensures consumer satisfaction and system’s health maintenance. Consumers usually submit a complaint or a request during power cut or while other issues persist.Tracking the amount of comments received and their type, along with the departments they fall under, time taken by the customer care team to reply will help in understanding the health of the distribution system across the company.

Any area that receives complaints on a regular basis or the ones that have a higher percentage of negative comments, needs more attention. It could be an area that is prone to theft or needs to repair/replace a portion of the system to better the quality of distribution of electricity. Different patterns and trends brought about by the analysis of consumer sentiment can help in proper maintenance and improvement of the same.

1. **LIBRARIES INCORPORATED**
2. **numpy**: Used for numerical operations on arrays and matrices.
3. **pandas**: Used for data manipulation and analysis, particularly with tabular data.
4. **re**: Used for regular expression operations for string matching and manipulation.
5. **warnings**: Used to handle and suppress warnings in the code.
6. **torch**: Core PyTorch library for building and training neural networks.
7. **os**: Used for interacting with the operating system, such as file and directory operations.
8. **sklearn.metrics import log\_loss**: Computes the logistic loss, which is useful for evaluating probabilistic classifiers.
9. **sklearn.feature\_extraction.text import TfidfVectorizer**: Converts a collection of raw documents to a matrix of TF-IDF (Term Frequency-Inverse Document Frequency) features.
10. **sklearn.linear\_model import LogisticRegression**: Implements logistic regression for binary and multiclass classification tasks.
11. **sklearn.naive\_bayes import MultinomialNB**: Implements the Multinomial Naive Bayes algorithm for classification, particularly effective for text data.
12. **sklearn.ensemble import RandomForestClassifier**: Implements a random forest classifier, an ensemble method that combines multiple decision trees for improved accuracy.
13. **sklearn.svm import SVC**: Implements Support Vector Classification, a powerful classifier for binary and multiclass tasks.
14. **sklearn.neighbors import KNeighborsClassifier**: Implements the k-nearest neighbors algorithm for classification tasks.
15. **sklearn.tree import DecisionTreeClassifier**: Implements a decision tree classifier, a model that predicts target values by learning simple decision rules from data.
16. **torch.nn as nn**: Provides neural network layers and operations for building models in PyTorch.
17. **torch.optim**: Contains optimization algorithms like SGD and Adam for training models.
18. **torchtext.vocab import GloVe**: Imports pre-trained GloVe word embeddings for use in NLP models.
19. **torch.utils.data import TensorDataset, DataLoader, random\_split**: Provides utilities for handling datasets and creating data loaders for training and validation.
20. **sklearn.model\_selection import train\_test\_split**: Splits data into training and testing sets.
21. **sklearn.preprocessing import LabelEncoder**: Encodes target labels with value between 0 and n\_classes-1.
22. **sklearn.feature\_extraction.text import CountVectorizer**: Converts text data into a matrix of token counts.
23. **torch.nn.functional as F**: Contains functions for neural network layers and operations (e.g., activation functions).
24. **tqdm**: Used for displaying progress bars for loops and other iterable operations.
25. **matplotlib.pyplot as plt**: Used for creating static, interactive, and animated visualizations in Python.
26. **transformers import AutoTokenizer, AutoModelForSequenceClassification**: Provides pre-trained tokenizers and models for sequence classification tasks using the Transformers library.
27. **sklearn.metrics import accuracy\_score**: Computes the accuracy classification score.
28. **transformers import BertTokenizer, BertForSequenceClassification**: Imports pre-trained BERT tokenizer and model for sequence classification tasks.
29. **collections import Counter**: Used for counting hashable objects, particularly for counting occurrences in a dataset.
30. **DATA PREPROCESSING**

The following pre-processing steps were used to ready the data: -

1. **Converted the customer feedback** to string datatype to ensure consistent text processing.
2. **Removed any rows** with None values to clean the dataset.
3. **Cleaned the text feedback** by removing newline characters, non-alphanumeric characters (except spaces), and extra whitespace, and by stripping leading and trailing whitespace.
4. **Filtered out feedbacks** that contained only numeric digits by replacing such entries with None.
5. **Filtered out very short feedback** by ensuring that each entry contained more than one word, removing entries with one or zero words.
6. **TRAINING OF MODELS**

I tried a bunch of different tools and models to achieve the main objective of Sentiment Analysis, these include various Machine Learning Models and Deep Learning Models.

* 1. **Machine Learning Models**

To convert text data into numerical data, in Machine Learning we use Vectorizers. This transformation is crucial because machine learning models require numerical input. Vectorizers analyze the text and create a matrix of features, where each feature represents a term or token from the text. I have used the following

**vectorizers**: -

1. **CountVectorizer**: CountVectorizer converts a collection of text documents to a matrix of token counts. It counts the number of occurrences of each word (token) in the documents and assigns that number to the word (token).
2. **TfidfVectorizer**: TfidfVectorizer converts the text into a matrix of TF-IDF (Term Frequency-Inverse Document Frequency) features. It first tokenizes the text and counts the occurrences of each word, similar to CountVectorizer. then applies a weighting scheme that considers not only the frequency of a word in a document (TF) but also the rarity of the word across the entire corpus of documents (IDF).

In machine learning to classify a type of input, we use classification techniques. I have used the following **classification techniques**: -

1. **Logistic Regression**: This classifier is used for classification problems. It models the probability of an instance belonging to one of the two classes based on its features.
2. **Multinomial Naive Bayes**: This classifier is commonly used for text classification tasks. It assumes that the features (words) are independent and calculates the probability of a document belonging to each class based on the frequency of words in the document.
3. **Random Forest**: This classifier is an ensemble learning method that combines multiple decision trees. It creates many decision trees from random subsets of the data and takes the average or majority vote of their predictions.
4. **Decision Tree**: This classifier builds a tree-like model where each internal node represents a decision based on a feature, and each leaf node represents a class label. It creates rules for classifying instances based on their feature values.
5. **Support Vector Machine (SVM)**: This classifier finds the best hyperplane (or decision boundary) that separates the classes in the feature space. It aims to maximize the margin between the closest data points of different classes.
6. **K-Nearest Neighbors (KNN)**: This classifier assigns a class label to a new instance based on the majority class of its k nearest neighbors in the feature space. The distance metric (e.g., Euclidean distance) determines the "nearness" of instances.
   1. **Deep Learning Models**

To convert text data into numerical data, in Deep Learning we use Word Embeddings. This transformation is crucial because neural networks require numerical input to process. A lot of different word embeddings are available online. The one which I used is GloVe.

**GloVe Embedding**: Global Vectors for Word Representation(GloVe) is a widely-used word embedding technique in natural language processing (NLP). It was developed by researchers at Stanford University and is designed to capture the semantic relationships between words in a corpus by representing them as dense vectors in a high-dimensional space.

After converting the text into numbers the crucial parts come to design a complex neural network for the classification task. I tried three different models for the same: -

1. **LSTM + Attention Based Model**: This models take advantage of the complex algorithms and techniques of Attention Mechanism and the Long-Short Term Memory (LSTM) Cell.

It follows the following sequence: -

**a.) Input**: Sequence of word indices.

**b.) Embedding Layer**: Converts input words into dense vectors.

**c.) LSTM Layer**: Processes the sequence of embeddings to produce hidden states.

**d.) Attention Layer**: Computes attention weights and produces a context vector.

**e.) Context Vector**: The result of applying attention to the LSTM outputs.

**f.) Dropout Layer**: Applies dropout to prevent overfitting.

**g.) Fully Connected Layer 1**: Transforms the context vector.

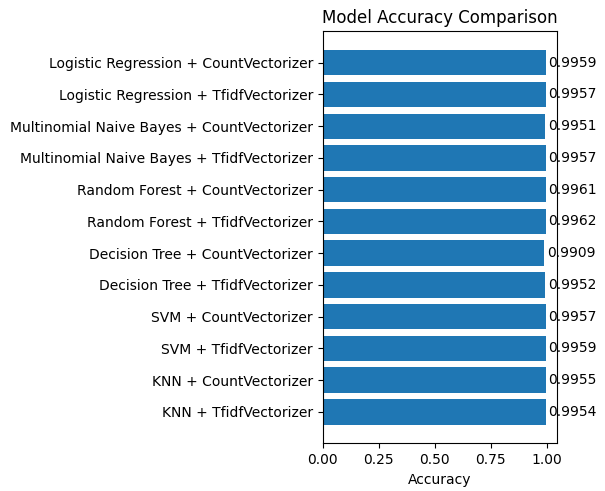
**h.) ReLU Activation**: Applies the ReLU activation function.

**i.) Fully Connected Layer 2**: Produces the final sentiment score.

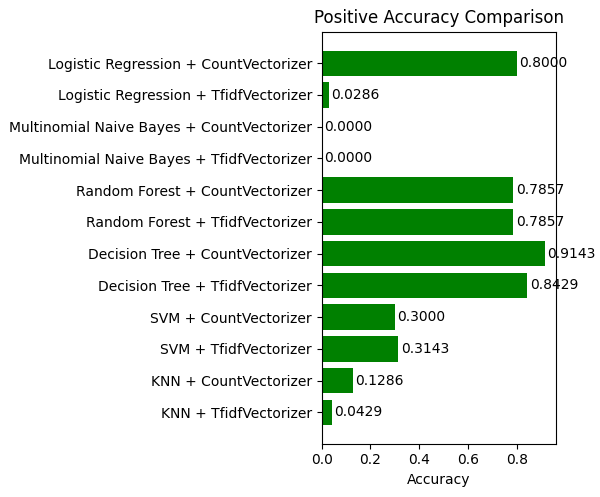
**j.) Output (Sentiment Score)**: The final sentiment classification result.

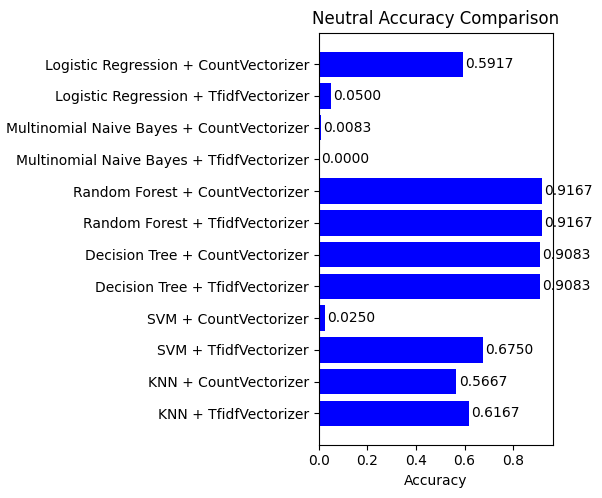
1. **XLM-RoBERTa Model**: This model is a specialized version of the RoBERTa (Robustly Optimized BERT Pre Training Approach) architecture, fine-tuned specifically for sentiment analysis on Twitter data. XLM-RoBERTa is a multilingual version of RoBERTa designed to handle multiple languages effectively. It is trained on a diverse set of languages, making it robust for multilingual sentiment analysis. It has been fine-tuned on a dataset of Twitter posts to classify them into sentiments (positive, neutral, negative).
2. **BERT Model**: This model BERT (Bidirectional Encoder Representations from Transformers) is fine-tuned specifically for sentiment analysis. The model is capable of handling multiple languages. It has been trained on text from various languages, making it effective for sentiment analysis across different languages. This model takes care of the uppercase and the lowercase letters as well. It has been fine-tuned on a dataset to classify text into different sentiment categories, typically positive, neutral, and negative.
3. **RESULTS**
   1. **Machine Learning Models**

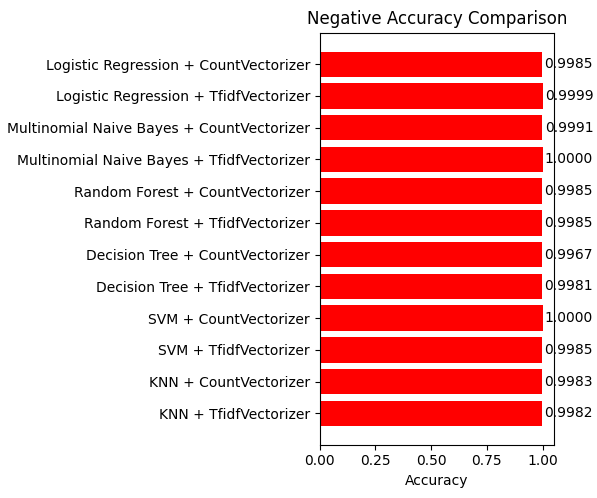
Below is the graph of final accuracies of all the models: -

****

As all of the models were giving unexpectedly very high accuracy, to further compare the performance of the models, I plotted the accuracy on the basis of classes (i.e. Positive, Neutral, Negative).





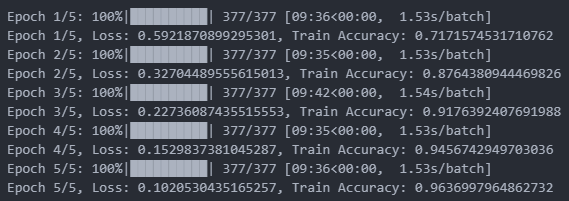


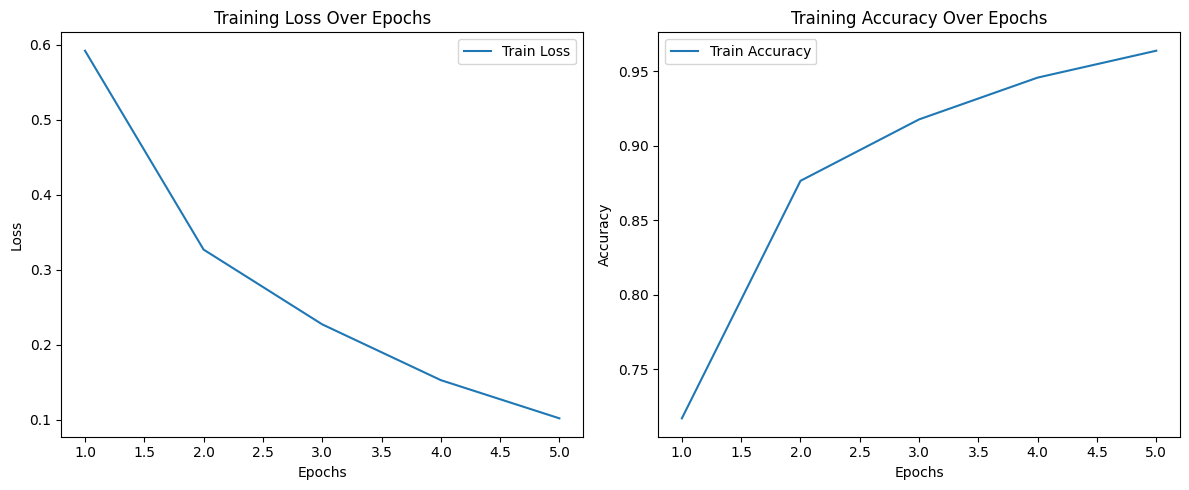
From the plots, the Decision Tree with CountVectorizer seemed to perform best. However, manual testing revealed its limitations in understanding word relationships. This led to the conclusion that Deep Learning models, which can capture these relationships, are better suited for sentiment analysis.

* 1. **Deep Learning Models**

1. **LSTM + Attention Based Model**

The following are the loss and accuracy plot while training the model.





Final Accuracy: 0.895 = 89.5%

1. **XLM-RoBERTa Model**

Final Accuracy: 0.783 = 78.3%

1. **BERT Model**

Final Accuracy: 0.857 = 85.7%

1. **DRAWBACKS**

As of 2024 there exists no such model that can effectively transcribe the comments received in hindi characters into english.The current sentiment analysis model can not utilize the Hindi comments for analysis purpose, but it uses custom model (LSTM+ATTENTION MODEL) to process words written in Hinglish ( Hindi comments written in english script).

Due to this drawback, the model automatically labels the hindi comments as neutral.

1. **FINAL PREDICTION MODEL**

Each of the deep learning models demonstrated strong performance individually. However, each model exhibited a bias towards certain types of input. To mitigate this issue and enhance overall accuracy, the final sentiment prediction model was developed as an ensemble, combining the strengths of all three models.

Accuracy of Final Model: 0.921 = 92.1%