Project Report: Sentiment Classification of Customer Reviews

1. Project Overview

This project involves classifying customer reviews as **positive** or **negative** using a machine learning approach. We use a publicly available dataset, perform text preprocessing, apply feature extraction, train a machine learning model, and evaluate its performance. The key goal is to develop a sentiment analysis model that can understand the sentiment behind customer reviews and classify them effectively.

2. Problem Statement

The objective of this project is to build a machine learning model that can automatically classify customer reviews as **positive** or **negative** based on their content. This is a typical **text classification** problem that requires handling and processing textual data effectively to produce a model that performs well in predicting sentiments.

---

3. Data Collection

For this project, a publicly available dataset of customer reviews was used. The dataset contains two columns:

- **Review Text**: The actual content of the customer review.
- **Sentiment**: The sentiment label (either "positive" or "negative") associated with the review.

Example of Dataset:

Review Text	Sentiment
"Great product, highly recommend!"	positive
"Very bad quality, not worth it."	negative

4. Data Preprocessing

Before training the model, the text data must be preprocessed to make it suitable for machine learning. The preprocessing steps include:

1. **Text Cleaning**:
    - Removed unnecessary characters, digits, and punctuation from the text.
    - Converted all text to lowercase to maintain uniformity.
2. **Tokenization**:
    - The cleaned text was split into individual words (tokens) to simplify analysis.
3. **Stopwords Removal**:

- Common words like “the,” “is,” “and,” which don’t add much value to the model, were removed from the text.

5. Feature Extraction

The next step was to convert the cleaned text into numerical format so that the machine learning algorithm could understand it. We used the **TF-IDF (Term Frequency-Inverse Document Frequency)** method for feature extraction. TF-IDF measures the importance of a word in a document relative to its frequency across all documents. This technique is preferred over the **Bag of Words** method because it reduces the impact of frequently occurring words that are common across all documents and highlights the more significant words in a given document.

- **Why TF-IDF**:
  - Unlike the Bag of Words model, TF-IDF assigns higher importance to words that are more relevant to specific reviews and reduces the weight of frequently occurring terms across all reviews.

6. Model Selection and Training

For the classification model, we selected the **Naive Bayes (MultinomialNB)** algorithm, which is widely used for text classification tasks. It is particularly effective for text-based problems because it assumes that the presence of a word in a review is independent of other words.

Training Process:

- **Data Split**: We split the data into training (80%) and testing (20%) datasets.
- **Model Training**: The Naive Bayes classifier was trained on the training data.
- **Prediction**: After training, the model was used to predict the sentiment of the test data.

7. Model Evaluation

The performance of the trained model was evaluated using the following metrics:

- **Accuracy**: Measures the overall correctness of the model, i.e., the percentage of correct predictions.
- **Precision**: Indicates the proportion of positive predictions that were actually correct.
- **Recall**: Measures the proportion of actual positive instances that were correctly identified by the model.

These metrics provide a comprehensive understanding of the model’s performance.

Example Evaluation Results:

- **Accuracy**: 88%
- **Precision**: 87%
- **Recall**: 89%

8. Hyperparameter Tuning (Optional)

To improve the model's performance, **hyperparameter tuning** was conducted using **GridSearchCV**. The primary hyperparameter tuned was the **alpha** value in the Naive Bayes classifier, which controls the smoothing effect. GridSearchCV helps to find the optimal value of this parameter by testing multiple values and selecting the one that provides the best performance.

Best Parameter:

- **Alpha**: 1

After tuning, the model showed slight improvements in accuracy, precision, and recall.

9. Model Improvement Strategies

Although the Naive Bayes model performed well, there are several ways to improve it further:

1. **Experiment with Different Algorithms**:
    - **Logistic Regression** and **Support Vector Machines (SVM)** can be tested for potentially better performance, especially when dealing with more complex data.
2. **Use More Data**:
    - Increasing the dataset size can help the model generalize better, reducing overfitting and improving performance on unseen data.
3. **Deep Learning Models**:
    - Advanced models such as **Recurrent Neural Networks (RNN)** or **Transformers (like BERT)** can be employed for better feature representation, especially in understanding the context of words in a sentence.
4. **Feature Engineering**:
    - Incorporating other features, such as review length or sentiment scores from external tools, may improve performance.
5. **Word Embeddings**:
    - Using **Word2Vec** or **GloVe** embeddings instead of TF-IDF could potentially capture the semantic meaning of words more effectively.

11. Challenges Faced

- **Data Cleaning**: Removing unnecessary characters and stopwords from the text can sometimes lead to the loss of important context.
- **Model Performance**: Naive Bayes is simple and efficient but may not capture the complexity of sentiments as well as deep learning models could.