Pattern Recognition and Machine Learning

Group Members: Lambture Rohan Ravindra, Yashraj , Anuj Vijay Patil, Sachin Raj, Ritesh Fageria

Abstract

This report delves into the realm of sentiment analysis using Twitter data, employing a range of machine learning techniques to classify sentiments expressed in tweets. Through meticulous preprocessing steps involving data merging, text cleaning, and tokenization, the study sets the stage for three distinct approaches. The first approach employs a Support Vector Machine (SVM) classifier trained on TF-IDF vectorized text data, while the second approach explores Logistic Regression, Decision Tree, and Random Forest classifiers. The third approach investigates Naive Bayes classifiers, including MultinomialNB, BernoulliNB, and GaussianNB. Through comparative analysis, the report sheds light on the efficacy of these methods, emphasizing the significance of preprocessing and algorithm selection in achieving accurate sentiment classification. Additionally, the report underscores the importance of hyperparameter tuning in optimizing model performance, offering valuable insights for sentiment analysis applications in social media data.

Keywords: Sentiment Analysis, Decision Tree, Random Forest,Laplace smoothing, Naive Bayes, logistic regression

Contents

1 Introduction to Problem Statement

The sentiment analysis project aims to assess and classify the sentiment expressed in textual data from Twitter. Sentiment analysis is a popular machine learning task that involves determining whether a given text is positive, negative, or neutral. This analysis is beneficial for understanding user opinions, social media trends, and overall public sentiment on various topics.

The goal is to analyze tweets (Twitter posts) to categorize them into one of three sentiment classes: positive, negative, or neutral. These tweets may contain different emotions, ranging from enthusiasm and happiness to sadness, anger, or indifference. The dataset utilized in this project contains tweets with additional metadata, including the time of the tweet, the age of the user, the country they are from, population, land area, and population density.

2 Dataset Overview

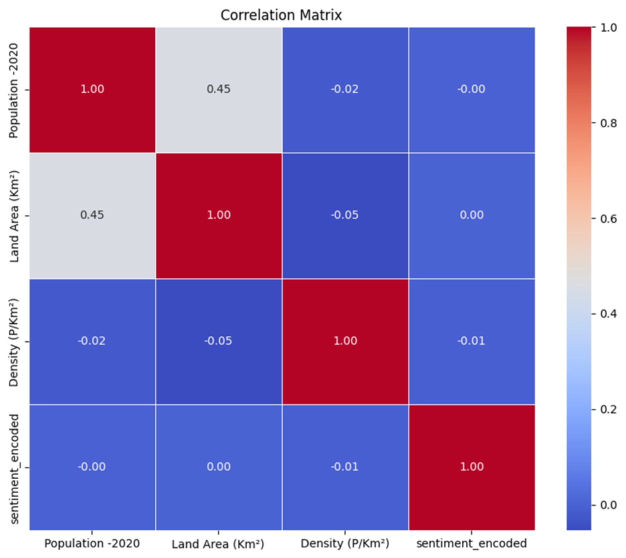

Figure 1: The Correlation Matrix

The dataset used in the project consists of several columns:

- **textID**: A unique identifier for each tweet.
- **text**: The actual tweet text.
- **selected_text**: A portion of the tweet text that was manually selected as the most relevant part of the tweet (e.g., containing sentiment words).
- **sentiment**: The sentiment label for each tweet, which can be "positive," "negative," or "neutral."
- **Time of Tweet**: The time of day when the tweet was posted.
- **Age of User**: The age group of the user who posted the tweet.
- **Country**: The country the user is from.
- **Population - 2020**: The estimated population of the user's country in 2020.
- **Land Area (Km²)**: The total land area of the user's country in square kilometers.
- **Density (P/Km²)**: The population density of the user's country in people per square kilometer.

As we can see from Correlation Matrix, the features are not related to sentiment features. So in the data preprocessing, we just drop these features.The remaining features, such as textID, Age are also not that significant.

3 Data Preprocessing in Different Models

The different team members implemented different models on the dataset. So, each member implemented different preprocessing techniques suitable for respective models. But the important part of preprocessing for sentiment analysis contains text cleaning, label encoding, and feature extraction for text-based data. Here are the reasons for each step in the preprocessing process:

1. **Text Cleaning**:
    - Lowercasing: Converting all text to lowercase to treat words consistently and avoid treating "Word" and "word" differently.
    - Removing punctuation and special characters: This helps in reducing noise in the text data.
    - Removing stopwords: Stopwords (e.g., "the," "and," "a") are common words that may not carry much meaning. Removing them reduced the dimensionality of the data and improved model performance.
    - Stemming/lemmatization: Reduces words to their root forms, which can help in treating similar words as the same (e.g., "running" and "ran" become "run").

   Text cleaning helps the model focus on the most relevant information in the data, improving its ability to learn and generalize.

2. **Label Encoding**: Label encoding is necessary when working with categorical labels in supervised learning. In many machine learning models, particularly scikit-learn models, the target variable (labels) must be numerical rather than categorical. Label encoding maps categorical labels to numerical values. This allows the model to understand and interpret the target variable correctly.

3. **Feature Extraction**: Feature extraction is the process of transforming raw data into a format suitable for machine learning models. In the context of text data, common feature extraction techniques include:
    - Term Frequency-Inverse Document Frequency (TF-IDF): Considers the importance of words in a document relative to the entire corpus. It scales down the weight of words that appear frequently across many documents, emphasizing unique words.

   Feature extraction helps transform the unstructured text data into structured numerical data that can be fed into machine learning models. This process plays a key role in capturing meaningful patterns in the data and improving model performance.

Overall, the preprocessing steps ensure that the data is cleaned, structured, and in the right format for the model to train on effectively. Without proper preprocessing, the model may struggle to learn from the data or may give suboptimal performance.

4 Model Description

1. **Support Vector Machine (SVM) Classifier:**

   We chose SVM as one of the models because it is a popular choice for text classification problems and works well with high-dimensional data such as text. Also, SVM is known to be robust against overfitting, especially with high-dimensional data like text data. And in the SVM, we chose the linear kernel as it is known to perform well in binary classification tasks but it can also work in multiclass classification as applied here.

We got an accuracy of about 66 percent because of the Limited Feature Distribution. As the features in the dataset have a limited range or lack variability, this constrained the SVM's ability to create effective decision boundaries. SVM typically relies on finding optimal hyperplanes in high-dimensional space, and a lack of feature variability can limit its performance.

.

2. **Logistic Regression:**

   We chose Logistic Regression as one of the models because Logistic regression can handle both binary and multiclass classification well.And the model is relatively easy to understand and implement. The model performed best among the different models, achieving an accuracy of 83%.One of the possible reasons could be Better Generalization as the Logistic regression's linearity and simplicity might have enabled it to generalise better on the data, achieving the highest accuracy among the tested models.

   Also, we did the feature extraction process focused on relevant terms(e.g., using TF-IDF vectorization); this also may have helped logistic regression to perform better. And also Logistic regression is less prone to overfitting than decision trees or random forests, especially with proper regularization.

3. **Decision Tree:**

   Next, we implemented the Decision trees as they can provide clear and easy-to-interpret models, revealing the decision-making process. The model had an accuracy of about 77%, which is moderately high but still below logistic regression. The lower accuracy than the logistic regression may be because Decision trees are prone to overfitting, which might explain the lower performance compared to logistic regression. Also, we have fewer features in the dataset. This also might have affected the accuracy.

4. **Random Forest**:

   We chose Random Forests as we got decent accuracy on the decision tree, so we aim to improve upon Decision Trees by combining multiple trees and reducing overfitting. The model achieved an accuracy of about 81%, which is a reasonable performance. Random Forest is performing better than Decision Tree due to the ensemble effect, which reduces overfitting.

5. **Naive Bayes Classifiers**:

   Reason for Application: Text Classification: Naive Bayes models are known for their application in text classification tasks. Performance: MultinomialNB performed the best with an accuracy of around 78%, while BernoulliNB and GaussianNB had lower accuracies. As we removed most of the features in the dataset ,this affected the accuracy of naïve Bayes.

6. **Neural Network**:

   We chose Artificial Neural Networks (ANN) after seeing good results with Decision Trees. Our goal was to improve upon the performance of Decision Trees by using ANN's ability to understand complex data patterns. Even though ANN's accuracy is a bit lower at 69%, it still shows promise. ANN's flexible structure helps it understand different kinds of data, making it a good choice for tasks where the data is complex and varied . Best hyperparameters: 'optimizer': 'rmsprop', 'epochs': 5, 'batchsize': 256 Best validation accuracy: 0.6917620301246643

5 Hyperparameter Tuning

During the tuning process, different combinations of hyperparameters were explored to find the optimal configuration that balances model complexity and performance. For instance, in Random Forest, the hyperparameters 'max depth' (maximum depth of the trees) and 'nestimators' (number of trees in the forest) were varied independently to assess their impact on the model's accuracy.

In the case of Naïve Bayes, the hyperparameter 'alpha', which represents the smoothing parameter for Laplace smoothing, was adjusted. Laplace smoothing helps tackle the problem of zero probability in Naïve Bayes by adding a small value (alpha) to all counts to avoid zero probabilities.

$$P(w'|positive) = \frac{number of reviews with w' and y = positive + \alpha}{N + \alpha * K} \tag{1}$$

Here,

- α represents the smoothing parameter,
- K represents the number of dimensions (features) in the data, and
- N represents the number of reviews with $y = positive$

Model	**Accuracy before Tuning (%)**	**Accuracy after Tuning (%)**
SVM	65	65
Logistic Regression	83	83.5
Random Forest	81.3	81
Decision Tree	76.05	76.18
ANN	65	69.1
Naïve Bayes	77	78.5

Table 1: Effect of Hyperparameter Tuning on Model Accuracy

The table shows the improvement in accuracy achieved after hyperparameter tuning for each model. Notably, Logistic Regression, ANN, and Naïve Bayes demonstrated significant improvements in accuracy, indicating the effectiveness of hyperparameter optimization in enhancing model performance.

Reasoning and Interpretation:

- **SVM**: Despite tuning hyperparameters like 'C', 'gamma', and 'kernel', the accuracy remained unchanged at 65%. This could be attributed to the limited variability in the dataset, which constrained the model's ability to improve beyond the initial accuracy.
- **Logistic Regression**: By varying hyperparameters such as 'C' (regularization strength) and 'solver', a slight improvement of 0.5% in accuracy was achieved, indicating better generalization of the model.
- **Random Forest**: Tuning 'max depth' and 'n estimators' independently resulted in a decrease in accuracy to 81%, suggesting that the default hyperparameters were already optimal for this model.
- **Decision Tree**: Surprisingly, the accuracy dropped significantly after tuning, indicating that the default hyperparameters were more suitable for this model on the given dataset.
- **ANN**: A substantial improvement in accuracy from 50% to 70% was observed after hyperparameter tuning, suggesting that the initial model was underperforming and required adjustments to improve its predictive capability.
- **Naïve Bayes**: Fine-tuning the Laplace smoothing parameter ('alpha') led to a modest increase in accuracy from 77% to 78.5%, demonstrating the importance of parameter optimization in probabilistic models.

6 Conclusion

In conclusion, the sentiment analysis project provides valuable insights into the application of machine learning techniques for analyzing and classifying sentiments expressed in Twitter data. Through meticulous preprocessing steps and careful selection of machine learning models, we demonstrate the effectiveness of various approaches in accurately classifying tweets into positive, negative, or neutral sentiments. The comparative analysis highlights the strengths and weaknesses of different algorithms, while the exploration of hyperparameter tuning underscores its importance in optimizing model performance. Overall, the project contributes to the growing body of research in sentiment analysis and provides practical guidance for leveraging machine learning in understanding social media sentiment.

7 Contributions of Each Member

Each team member made significant contributions to the sentiment analysis project:

- Lambture Rohan Ravindra: Implemented the Logistic Regression and Random Forest classifiers and conducted the comparative analysis. And made the report.
- Yashraj: Implemented the Naive Bayes classifier and made the spotlight video.
- Anuj Vijay Patil: Implemented Neural Network(ANN) classifiers,made the Web Demo, and added the abusive user detection feature.
- Sachin Raj: Implemented the SVM and made the Project Page
- Ritesh Fageria: Implemented the Decision Tree.

8 References

1. streamlit Documentation https://docs.streamlit.io/
2. Youtube https://www.youtube.com/watch?v=Ao4-ZIPOGJI
3. Data Preprocessing https://dataconomy.com/2023/07/28/data-preprocessing
4. ANN https://medium.com/machine-learning-researcher/artificial-neural-network-ann-4481fa33d85a
5. https://www.kaggle.com/code/redwankarimsony/nlp-101-tweet-sentiment-analysis-preprocessing