Binary text classification

Name: Nasim Bayat Chaleshtari

Student Number: 20240207

Introduction:

This report addresses the challenge of binary text classification, where the goal is to classify documents based on a term frequency matrix (data_train.npy) and binary labels (label_train.csv). A significant challenge in this project is the **class imbalance**, with a much larger number of documents labeled as class 0 compared to class 1. Additionally, the raw term frequency vectors require preprocessing to focus on the most relevant features for accurate classification.

To address these challenges, I followed a multi-step approach. Initially, I applied **TF-IDF transformation** to convert the term frequency matrix into weighted features, and used **feature selection** techniques such as **chi-squared** and tried **PCA** to reduce dimensionality (But has not good impact) and retain only the most informative features. I experimented with several machine learning models, including **Logistic Regression**, **SVM**, **LightGBM**, **Naive Bayes**, **Xgboost**, **Random forest, K-nearest neighbors** and a **Neural Network**, incorporating strategies like **class weighting** to handle the class imbalance.

I also explored **ensemble methods** like **Voting Classifier** and **Stacking** to improve performance and use power of different algorithms for prediction. Among these,

Voting classifier with combination of LogisticRegression, BernouliNB, SVC and light gbm classifier has highest Macro f1 score with (**0.738)** and for indivisual models **Logistic Regression** achieved the best performance with a **Macro F1 score of (0.733)**, followed closely by **SVM** (**0.723**) and **LightGBM** (**0.721**). The **Neural Network** performed relatively lower, with a **Macro F1 score of (0.65)**, though it showed a **79% accuracy** on the validation set. The results indicate that while more complex models like the Neural Network had high accuracy as far as I read in websites, they struggled to effectively balance both classes in the classification task.

In summary, the best results came from Voting classification and after that **Logistic Regression model**, which provided a good balance between performance and interpretability, and a range of different techniques were tested to improve classification accuracy and handle the challenges of the dataset.

For fist task which we should not use any machine learning libraries, I write naïve bayse algorithm by myself and it has very good result with 79% accuracy on my validation data and 74% f1 micro average on public test in Kaggle, but for private test data this algorithm has 71.96% f1 micro. but Logistic regression model has 72.477 and voting classifier reach to 72.958 which shows that Logistic model and Voting classifier with combination of several algorithm better coverage unseen data.

Feature Design

The feature design process for this project focused on transforming the raw term frequency data into a more meaningful and usable format for classification. Several pre-processing and feature extraction techniques were applied to enhance model performance and ensure that only the most relevant features were used in training. The key steps involved were **TF-IDF transformation**, **feature selection**, and **handling class imbalance**.

1- **TF-IDF Transformation**:
   - **Raw term frequency counts** were initially represented as sparse vectors in data_train.npy, where each entry indicates the frequency of a specific term in a given document. These term frequency vectors alone can be ineffective because they do not capture the importance of terms across the entire dataset.
   - To address this, the **TF-IDF (Term Frequency-Inverse Document Frequency)** transformation was applied using TfidfTransformer from **scikit-learn**. This transformation adjusts the term frequencies by weighing terms based on their inverse document frequency, which reduces the impact of common terms (e.g., "the", "is", "and") and highlights more informative, rare terms.

- The TF-IDF approach helps emphasize terms that are discriminative for the target classification, making the feature representation more relevant and informative for the model.

2- **Feature Selection**:
    - With the transformation into TF-IDF features, the dataset remained high-dimensional, which could lead to overfitting and increased computational complexity. To address this, **feature selection** was performed using the **Chi-squared test** with SelectKBest to select the top 10,000 most relevant features.
    - The Chi-squared test measures the association between each feature (term) and the target variable (label). By selecting the most statistically significant features, this step ensures that only the most relevant terms are retained, reducing noise and improving model efficiency.
    - The number of features (k=10,000) was chosen through experimentation, balancing the trade-off between maintaining enough information for the model to learn effectively and reducing the dimensionality to avoid overfitting.
    - Also I tried PCA algorithm for reducing dimensionality but it has not as good result as K-best approach.
3- **Handling Class Imbalance**:
    - Given the imbalanced nature of the dataset, where the majority of instances belong to class 0 and only a small portion to class 1, special attention was paid to handling the **class imbalance** during the model development process.
    - For models like **Logistic Regression**, **SVM**, and **LightGBM**, the **class_weight='balanced'** parameter was used. This automatically adjusts the weight of each class in the loss function, giving more importance to the minority class (label 1). This helps mitigate the bias towards the majority class, encouraging the model to learn to classify both classes more equally.

- Also I tried another technique for balancing my data which was **SMOTE** from imblearn package and oversampling method, but setting hyper parameter in models for imbalance data set had better impact.

Algorithms

In this project, several machine learning algorithms were employed to tackle the binary text classification task. The primary focus was on models that can handle high-dimensional, sparse data and address the challenges posed by class imbalance. The following algorithms were considered:

1- **Logistic Regression**:
   - Logistic Regression is a simple yet effective linear model for binary classification. It models the probability of a sample belonging to a particular class using a logistic function. In this project, **class weighting** (class_weight='balanced') was used to handle the class imbalance, ensuring that the model gave appropriate importance to the minority class this hyper parameter has significant impact on result.
   - Regularization was applied to prevent overfitting, with the **L2 regularization** term controlled by the hyperparameter **C**. The regularization strength was tuned to optimize model performance.
2- **Support Vector Machine (SVM)**:
   - The Support Vector Machine (SVM) is a powerful classifier that works well in high-dimensional spaces. For this project, a **linear kernel** was used, as it is well-suited to document classification tasks, where the data tends to be sparse and high-dimensional.
   - Like Logistic Regression, the **class_weight='balanced'** parameter was used to mitigate class imbalance, and the **C** parameter was tuned to adjust the regularization strength. SVM is known for its robustness in cases of small to medium-sized datasets, and its ability to handle large feature spaces efficiently.
3- **LightGBM (LGBM)**:
   - LightGBM is an efficient gradient boosting framework that is particularly well-suited for large datasets and high-dimensional feature spaces. It builds an ensemble of weak decision trees

iteratively, where each new tree corrects the errors of the previous ones.

- LightGBM was configured with **class weighting** to address the class imbalance issue, and hyperparameters like **learning rate**, **num_leaves**, and **n_estimators** were tuned to optimize the model's performance.
- The ability of LightGBM to handle sparse data and its robustness against overfitting make it a valuable model for this type of classification task.

4- Neural Networks:

- A simple Feedforward Neural Network (FNN) was also implemented for classification. The neural network consists of multiple layers of neurons, where each layer learns increasingly abstract representations of the input features.
- Regularization techniques, such as **dropout** and **L2 regularization**, were applied to prevent overfitting, especially given the high-dimensional nature of the data.
- Neural networks are particularly well-suited for capturing complex, non-linear patterns in data especially with using activation functions like ReLU, and they can generalize well when regularization is properly applied.

Each of these algorithms was tested using the **Macro F1 score** to evaluate their performance on the validation set. This allowed us to assess how well the models handled both the majority and minority classes. The performance of these algorithms was compared to determine which one best suited the dataset and task at hand.

Methodology

The approach to building and optimizing the classification model involved several key decisions related to data splitting, regularization, optimization, and hyperparameter tuning. These choices were aimed at improving the model’s

generalization ability, preventing overfitting, and addressing the challenges of class imbalance and high-dimensional data.

Training and Validation Split:

- The dataset was split into a training set and a validation set using an 80/20 ratio, with 80% of the data used for training and 20% for validation. This split was performed using train_test_split from scikit-learn, ensuring a random distribution of samples across both sets.
- The **training set** was used to train the model and fine-tune hyperparameters, while the **validation set** was used to assess the model's performance and ensure it generalized well to unseen data.

Preprocessing and Feature Engineering

The raw data consisted of document-wise term frequency vectors, which were transformed into TF-IDF features. This step was necessary because raw term frequencies tend to overemphasize common terms and fail to capture the importance of less frequent but highly relevant words. The TF-IDF transformation was done using `TfidfTransformer` from `sklearn.feature_extraction.text`, with `L2` normalization applied and sublinear term frequency set to False.

To further improve model performance, I employed feature selection to reduce the dimensionality of the term vectors and retain only the most relevant features. Specifically, I used **chi-squared test** with **SelectKBest** to select the top **10,000 features** based on their relevance to the target labels. I also experimented with **Principal Component Analysis (PCA)** and **Truncated SVD** as alternative dimensionality reduction techniques to see if they would yield better results. However, the **chi-squared** approach was the most effective.

Regularization Strategy

Given the high dimensionality of the dataset and the risk of overfitting, regularization techniques were incorporated into several models to improve generalization.

1- **Logistic Regression** and **SVM** models utilized **class_weight='balanced'** to address the class imbalance and prevent the model from being biased towards the majority class. Regularization in the form of **L2 penalty** was used to penalize large weights in both models, promoting simpler models that avoid overfitting.
2- For the **Neural Network** model, regularization techniques like **Dropout** and **L2 regularization** were applied. Dropout helps prevent overfitting by randomly deactivating certain neurons during training, forcing the model to learn more robust features. The **L2 regularization** penalizes the model for excessively large weights, promoting simpler models. The dropout rate was set to **50%** in each hidden layer to strike a balance between regularization and model capacity.

Optimization Tricks

To optimize model performance and speed up training, the following optimization tricks were employed:

1- **Early Stopping (Neural Network):** For the **Neural Network**, **early stopping** was employed to prevent overfitting. Training was halted when the validation accuracy no longer improved, which ensured that the model did not continue to learn noise or unnecessary details from the data. Early stopping was set with a **patience of 10** epochs, meaning that training would stop if the validation accuracy didn't improve for 10 consecutive epochs.
2- **Hyperparameter Tuning:** I used **Optuna** to optimize the hyperparameters for both **Logistic Regression** and **SVM**. Key hyperparameters like **C (regularization strength)**, **max_iter (number of iterations)**, and **solver** for Logistic Regression, and **C (regularization strength)**, **kernel type**, and **gamma** for SVM, were tuned. This allowed for an automated and efficient search for the optimal hyperparameters within a specified range. I used **cross-validation** (3-fold) within **Optuna** to evaluate each set of hyperparameters based on the **Macro F1 score**.

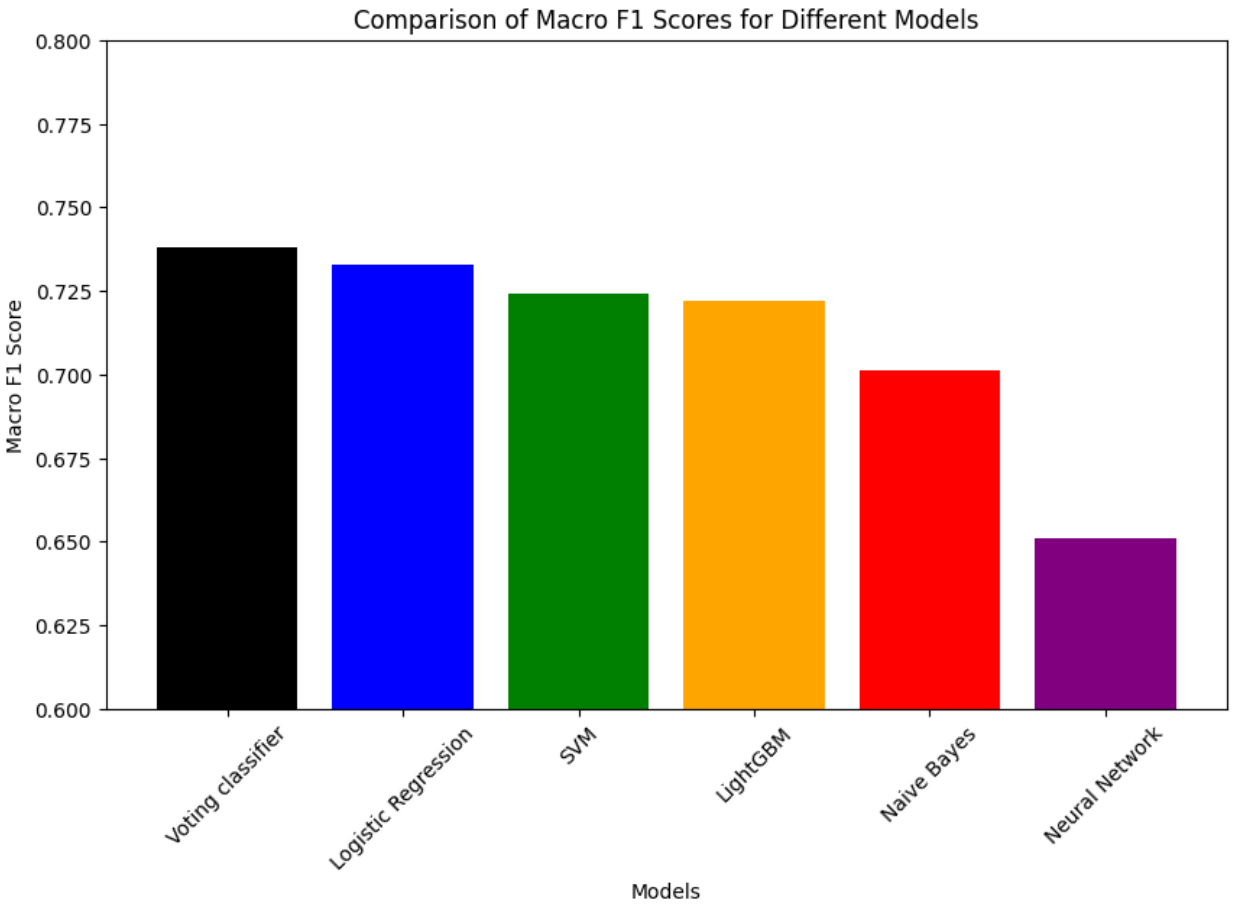
3- **Batch Size and Learning Rate (Neural Network):** For the Neural Network, the **learning rate** was set to **0.001** using the **Adam optimizer**, which adapts the learning rate during training for faster convergence. I also experimented with different **batch sizes**, ultimately settling on **64** based on performance during training. These settings helped the network learn efficiently while avoiding large, unstable updates to the weights.

Model Evaluation

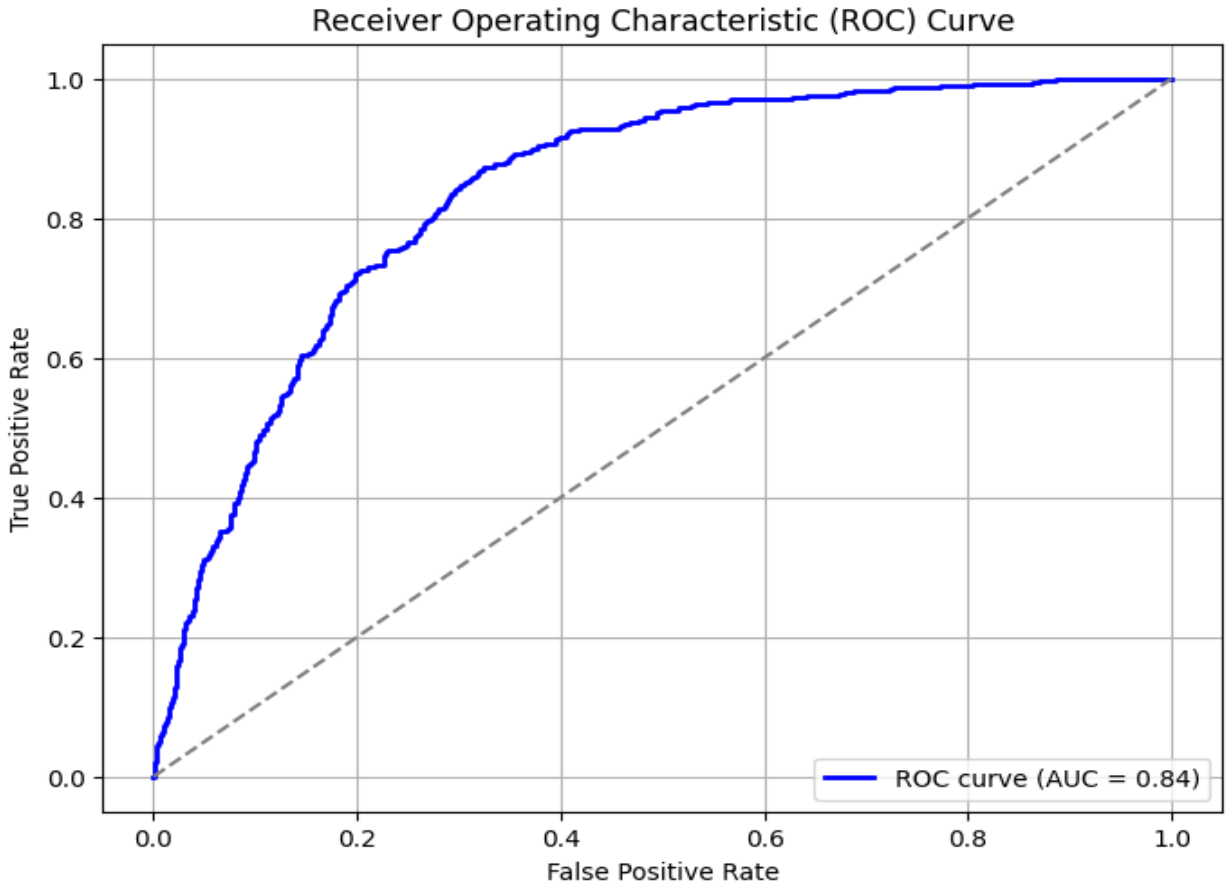
Once the models were trained, I evaluated them using the Macro F1 score, which provides a better sense of how well the model balances both classes (i.e., class 0 and class 1). The Macro F1 score was chosen because it provides an equal weight to both classes, making it particularly useful for imbalanced datasets. In addition to F1 scores, I also monitored accuracy and AUC (Area Under the Curve) to evaluate overall model performance.

Results

As I earlier mentioned Voting classifier and after that Logistic regression model has highest f1 micro average values compare to other models. I show the comparison between different models in below image.



Also another metric which is used for evaluation classification models is ROC AUC which I show the result of this metric for my best individual (Logistic regression) model in below image.



This plots show us the power of Logistic regression model with just a few hyper parameter tuning. It can classify well unbalance dataset with using very low resource and very faster rather than other algorithms like neural networks or Svm or other ensemble models. If I want to mention hyper parameters which has the most impact to training my model Is passing class weight as balance in logistic regression and after that the value of C was very important. Also a technique which help to improve model result was using TfidfTransformer with l2 normalization, I also tried l1 normalization but for our dataset l2 had better performance.

Discussion

In future maybe I tried LSTM layers for classification which I read about them very powerful in classification specially in long term dependency like texts data.

In milestone one I wirted naïve bayes algorithm by numpy library, this algorithm also is very fast in learning, but interesting note about this model was when I preprocessed data and train model by feature engineered data it has lower result on validation set rather than passing the raw X_train values to model. But for predicting Kaggle test dataset specially for private test data it has not as good as Logistic regression model performance with about 1% difference.

references

I used a lot formulas and contents in scikit-learn website

reading about how treat with imbalance dataset

writing neural network for binary classification

voting classifier

select k-best features