313689804 ;207653312

Report

Preprocessing Steps

1. Conversion of Device Type to Binary Values: The dataset contained a column indicating the device type ('android' or other). This was converted into a binary feature, where 'android' was labeled as 0 and other devices as 1. This transformation facilitates the use of this feature in machine learning models by encoding it as a numeric value.
2. Timestamp Handling: Timestamps were initially in string format. They were converted to numerical values representing the date in YYYYMMDD format. This transformation was done to normalize the temporal aspect of the data and to ensure the model can utilize time-related features effectively. Min-max scaling was applied to the transformed timestamp values to keep them within a standard range.
3. Tokenization of Tweet Text: Using the TweetTokenizer, each tweet's text was tokenized into individual words, ignoring case, and stripping out Twitter handles. This process breaks down the text into meaningful units that can be further analyzed and used for feature extraction or embedding.

Feature Engineering

1. Textual Features: Several features were extracted from the tweet text, including:
    - Uppercase Letter Counts: The number of uppercase letters in a tweet, which could indicate emphasis or shouting.
    - Hashtag Counts: The number of hashtags used, potentially indicating topic relevance or popularity.
    - Punctuation Counts: The number of punctuation marks, which might correlate with the sentiment or tone of the tweet.
    - Tag Counts: The number of times other users are tagged, which might reflect the tweet's reach or engagement level.
    - Token Length: The number of tokens in the tweet, providing a measure of its length.
    - Digit Counts: The number of digits present, possibly indicating references to numbers or statistics.
    - URL Counts: The frequency of URLs, often used to share external content or references.

These features were also scaled using min-max scaling.

2. Embedding-Based Features: For each token in the tweet text, embeddings were created using a pre-trained GloVe model. This allowed for capturing semantic information from the text. From these embeddings, additional features were derived:
    - Summed Embeddings: Sum of all token embeddings in a tweet.
    - Averaged Embeddings: Average of all token embeddings in a tweet.
    - Sum/Avg Ratio and Avg/Sum Ratio: Ratios calculated from summed and averaged embeddings, providing normalized measures of the tweet's semantic density.

These features were also scaled using min-max scaling to standardize their ranges.

The preprocessing and feature extraction steps were essential for optimizing the data for predictive modeling. Tokenizing tweet text and extracting features like uppercase and hashtag counts provided useful indicators for content characteristics. Embedding-based features, using pre-trained GloVe embeddings, added semantic depth to the analysis. These steps aimed to clean, normalize, and enrich the data, making it more suitable for machine learning tasks. Future work could include additional text processing techniques and exploring different embedding models.

Results

The models evaluated include Logistic Regression, Support Vector Machine (SVM), Feedforward Neural Network (FFNN), Random Forest, and a fine-tuned BERT model.

We present the accuracy results here, along with precision, as the data is not balanced, and we wanted to support additional analysis. Additional metrics such as recall, F1 score, and the confusion matrix are available in the codebook. These metrics were also crucial in helping us draw our conclusions.

Note that we performed a GridSearchCV to identify the optimal parameters for the model.

Model	Hyperparameters	Accuracy	precision
Logistic Regression	C: 10, Solver: newton-cg	0.78	0.76
Support Vector Machine	C: 100, Gamma: scale, Kernel: rbf	0.79	0.77
Feedforward Neural Network	Layers: [Input: 909, Hidden1: 256, Hidden2: 128, Hidden3: 32, Hidden4: 16, Output: 1], Activation Function: ReLU, Dropout: 0.3 , Epochs: 32, Loss Function: Binary Cross-Entropy , Optimizer: Adam (learning rate = 0.001)	0.74	0.69
Random Forest	Criterion: entropy, Max Depth: 4, Estimators: 256	0.78	0.76
BERT Fine-Tuned	Epochs: 7, Loss Function: Cross-Entropy Loss, Optimizer: Adam-default learning rate	0.873	0.86

Results and Analysis

Logistic Regression

This model is often favored for its simplicity and interpretability, particularly in scenarios with linearly separable data. Its moderate success in this task indicates that the preprocessing steps provided a feature set that allowed the linear decision boundaries of the model to capture significant patterns in the data.

Support Vector Machine (SVM)

SVM has ability to manage non-linear relationships with the rbf kernel. The tuning of hyperparameters (C: 100 and Gamma: scale) significantly contributed to its accuracy, demonstrating the model's adaptability to complex data patterns. However, despite its capacity to handle complex relationships, the SVM's performance was like that of the Logistic Regression model. This similarity might be due to the feature set being largely linearly separable, which limited the SVM's advantage in capturing non-linear patterns.

Feedforward Neural Network (FFNN)

The FFNN, with its layered architecture and ReLU activation function, did not perform as well as the other models. This relatively lower accuracy could be due to the model's complexity and the potential for overfitting, which the dropout layer (0.3) aimed to mitigate. The model's performance suggests that it may require more extensive data or further hyperparameter tuning to realize its full potential.

Random Forest

The Random Forest model, known for its ability to capture complex interactions. its performance was like simpler linear models like Logistic Regression. This may be due to the model's limited depth (Max Depth: 4) and potential overfitting, which could prevent it from generalizing well. Additionally, the feature space may not have been complex enough to fully utilize the Random Forest's capabilities. This suggests that the dataset was better suited to simpler models or that the Random Forest's hyperparameters might need further tuning.

BERT Fine-Tuned

The fine-tuned BERT model outperformed all other models. BERT's ability to understand and leverage contextual information in text through its transformer architecture gives it a distinct advantage in tasks requiring nuanced language understanding. The fine-tuning process, although only for 10 epochs, significantly enhanced its performance, indicating that BERT's pretrained weights were well-suited to the nuances of the dataset.

Conclusion

The significant performance differences across models underscore the critical role of model selection in machine learning tasks. The BERT model’s superior performance highlights the advantage of advanced language models in processing text data, especially when deep contextual understanding is essential. Conversely, simpler models like Logistic Regression and SVM also achieved notable results, suggesting that the dataset's features may have had some degree of linear separability or that SVM's kernel functions provided effective transformations. The FFNN and Random Forest models yielded moderate results, indicating that their increased complexity did not necessarily lead to improved performance. This observation suggests that, for this dataset, either a well-engineered simple model or a sophisticated pre-trained model like BERT is preferable.