Comparative Study of Algorithms used in predicting the virality of Reddit Threads

Brinda Shyamasundar, Adishree Sane, Sharvari Doijode, Nithyashree KJ

April 18, 2025

1 Introduction

This project focuses on predicting whether a Reddit post will go viral using machine learning. A post is labeled viral if it gains 300 or more upvotes. The dataset, collected via data scraping, includes features like subreddit, post title, number of comments, upvotes, and timestamp. We implemented Logistic Regression and XGBoost models, comparing their performance on various metrics.

2 Data Collection and Preparation

2.1 Data Scraping

Reddit data was scraped using the Pushshift API, which provides access to historical Reddit posts. We filtered data for multiple subreddits to ensure diversity. Each data entry included:

- **subreddit**: The community in which the post appeared.
- **title**: The textual content of the post.
- **upvotes**: Total number of upvotes.
- **comments**: Total number of comments.
- **timestamp**: Date and time the post was made.

2.2 Data Cleaning

Removed posts with missing values. Converted timestamps into datetime objects for future feature engineering. Dropped duplicate posts.

3 Exploratory Data Analysis (EDA)

The EDA phase aimed to understand the relationship between features and virality.

- **Distribution of Upvotes**: Highly skewed; most posts have ¡100 upvotes.
- **Subreddit Analysis**: Some subreddits naturally generate more engagement.
- **Correlation Analysis**: Positive correlation between number of comments and upvotes.
- **Temporal Patterns**: Posts during weekends and evenings received higher engagement.
- **WordClouds & Text Frequency**: Extracted frequent words from viral vs non-viral posts; emotional or controversial words were more common in viral posts.

Plots used: histograms, boxplots, violin plots, time series plots, and word clouds.

4 Feature Engineering

4.1 Target Variable

We defined the target viral as:

$$\text{viral} = \begin{cases} 1 & \text{if upvotes} \geq 300 \\ 0 & \text{otherwise} \end{cases}$$

4.2 Text Features

TF-IDF Vectorization on post titles. Tested n-gram ranges (1,1) and (1,2). Removed stopwords, lowercased all text.

4.3 Categorical Features

Label encoded subreddit. One-hot encoding discarded due to memory issues.

4.4 Numeric Features

Comments scaled using StandardScaler. Timestamp was parsed to extract hour and weekday.

4.5 Final Feature Matrix

Combined TF-IDF matrix with subreddit encoding and numeric features.

5 Logistic Regression Model

5.1 Overview

Logistic Regression calculates the probability of a binary outcome using:

$$P(y = 1|X) = \frac{1}{1 + e^{-(w^T X + b)}}$$

5.2 Loss Function

Binary Cross-Entropy Loss:

$$L = -[y \log(p) + (1 - y) \log(1 - p)]$$

5.3 Training

Used liblinear solver with L2 regularization. Handled imbalance using class_weight='balanced'. Cross-validation (5-fold) for hyperparameter tuning.

5.4 Results

- **Training Accuracy**: 0.87
- **Test Accuracy**: 0.61
- **F1 Score**: 0.65

Logistic Regression provided a good baseline but struggled with generalization to unseen data.

6 XGBoost Model

6.1 Overview

XGBoost is a tree-based ensemble model optimized for speed and performance. It uses boosting to minimize prediction error iteratively.

6.2 Objective Function

$$\text{Obj} = \sum l(y_i, \hat{y}_i) + \sum \Omega(f_t)$$

Where l is log loss and Ω is the regularization term (depth, leaves).

6.3 Boosting Process

Uses both gradient and Hessian (second-order) information. Successive trees fit residuals of previous ones. Controlled overfitting via regularization.

6.4 Hyperparameter Tuning

Tuned max_depth, eta, lambda, alpha, and subsample. Employed grid search and early stopping based on validation loss.

6.5 Results

- **Training Accuracy**: 0.99
- **Test Accuracy**: 0.74
- **F1 Score**: 0.75

XGBoost outperformed Logistic Regression on the test set, offering better handling of complex feature interactions and class imbalance.

7 Model Comparison

Model	**Accuracy (Train)**	**Accuracy (Test)**	**Precision**	**F1 Score**
Logistic Regression	0.87	0.61	0.70	0.65
XGBoost	0.99	0.74	0.77	0.75

XGBoost showed better performance on the test set, particularly in recall and F1 score, crucial for identifying viral posts. Logistic Regression offered interpretability and faster computation, but struggled with generalization.

8 Visual Analysis

Confusion matrices to analyze TP, FP, TN, FN. ROC curves and AUC scores to evaluate performance. Precision-Recall curves emphasized class imbalance.

9 Conclusion

Logistic Regression is interpretable and fast but limited in complexity. XGBoost captured nonlinearities and interactions, offering superior results, especially in recall. TF-IDF played a major role in improving accuracy.

10 Dependencies

- Python 3.8+
- pandas
- numpy
- scikit-learn

- xgboost
- matplotlib
- seaborn
- wordcloud
- datetime

LightGBM Model Pipeline

This section provides an in-depth explanation of the modeling process, data handling, and LightGBM configuration used in the notebook `lightbgm.ipynb`.

1. Data Preprocessing

1. **Column Pruning**
   A list of non-informative or redundant columns was dropped before modeling. These typically included:
   - Metadata: `approved_by`, `archived`, `author`, etc.
   - Identifiers/URLs: `post_url_from_permalink`, `permalink`, etc.
   - Columns with excessive missing or constant values
2. **Missing Value Handling**
   Missing values in selected features (e.g., `author_cakeday`) were filled with `None` or other default indicators.
3. **Feature Selection**
   Features were selected based on their relevance and data type, excluding sparse or irrelevant categorical fields.
4. **Data Splitting**
   The dataset was split using:

   ```
   train_test_split(X, y, test_size=0.2, random_state=42)
   ```
5. **Feature Scaling**
   Although LightGBM is scale-invariant, `StandardScaler` was applied to ensure compatibility with potential post-processing steps.

2. LightGBM Configuration

LightGBM is a gradient boosting framework based on decision tree algorithms, optimized for performance and scalability.

- **Model Used**: `lightgbm.LGBMRegressor`
- **Key Parameters**:

```
lgb.LGBMRegressor(
    n_estimators=1000,
    learning_rate=0.05,
    objective='regression',
    random_state=42
)
```

- **Training**: The model was trained with early stopping and validation monitoring using the `.fit()` method.

3. Evaluation Metrics

Model performance was assessed using standard regression metrics:

- **Root Mean Squared Error (RMSE)**:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

- **Coefficient of Determination ($\mathbf{R^2}$ Score)**:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

- **Example Output** (hypothetical):

```
RMSE: 0.4352
R² Score: 0.8721
```

4. Strengths of the Approach

- Efficient training via histogram-based learning in LightGBM
- Automatic handling of missing values and categorical splits
- Scalable to large datasets
- Straightforward integration with SHAP and feature importance tools

5. Potential Improvements

- **Hyperparameter Tuning**: Use `GridSearchCV` or `optuna` for tuning `max_depth`, `num_leaves`, etc.
- **Feature Engineering**: Add domain-relevant features, such as time-based attributes or sentiment scores.
- **Categorical Encoding**: Improve categorical variable handling with more explicit encoding or data typing.

[11pt]article [utf8]inputenc amsmath graphicx hyperref geometry margin=1in

CNN and LSTM model

article [utf8]inputenc geometry hyperref margin=1in Reddit Post Virality Prediction

Overview

This project focuses on predicting the virality of Reddit posts based on engineered features extracted from the post metadata. The dataset includes a diverse range of features that capture linguistic, temporal, and engagement-related patterns that potentially influence a post's popularity.

Data Preprocessing

The preprocessing step involved feature engineering to derive meaningful attributes from raw data. A total of 15+ features were generated, including:

- **Title and Content Length** — Number of characters in title and post body.
- **Sentiment Scores** — Compound sentiment scores for both title and content using VADER sentiment analysis.
- **Upvotes and Gilding** — Direct measures of post engagement.
- **Engagement Velocity** — Rate of upvotes over time.
- **Popularity Index** — A derived metric combining upvotes and gilding over time.
- **Post Timing Features** — Hour of posting, day of the week, and month extracted from timestamps.
- **Media Presence and Crossposts** — Binary features identifying media content and whether the post was crossposted.
- **Upvote to Gilded Ratio** — A proxy for quality or appreciation.

Virality Classification

A binary classification label named `virality_factor` was introduced, defined as:

> *Posts with more than 30 upvotes are considered* ***viral*** *(1), others are* ***non-viral*** *(0).*

The distribution of this label was visualized to analyze class imbalance and ensure appropriate modeling strategy.

Model Evaluation

The analysis highlights the performance of three models—Random Forest, LSTM, and CNN—in predicting the virality of posts:

- **Random Forest:** Achieved perfect performance metrics: 100% accuracy, precision, recall, F1-score, and an AUC-ROC of 1.0. Such flawless results suggest the model may be overfitting and should be validated on unseen data.
- **LSTM:** Designed for sequential data, this model reached 94.51% accuracy and shows potential for improvement with hyperparameter tuning.
- **CNN:** Obtained 91.30% accuracy and an AUC-ROC of 0.95, but recall was relatively low at 59.88%, indicating challenges in identifying all viral posts.

Feature Importance

Feature importance analysis using the Random Forest model revealed:

- Most significant predictors: `upvotes`, `popularity_index`, `upvote_to_gilded_ratio`.
- Less significant features: `title_length`, `selftext_length`, and sentiment scores.

This indicates that quantitative engagement metrics have a greater impact on virality prediction than qualitative features.

Recommendations

- **Validate Random Forest:** Test on unseen data to confirm generalizability and mitigate overfitting.
- **Address Class Imbalance:** Balance the dataset to improve recall, particularly for CNN.

- **Feature Exploration:** Include user-level features, subreddit behavior, and temporal trends.
- **Model Optimization:** Tune LSTM and CNN models or explore alternative architectures suited to tabular data.

Dependencies

The following libraries were used:

- `pandas, numpy, nltk, textblob, vaderSentiment`
- `matplotlib, scikit-learn`

BERT Model

In this project, we employed the **BERT (Bidirectional Encoder Representations from Transformers)** model to tackle a text classification problem. BERT has proven highly effective for various Natural Language Processing (NLP) tasks due to its contextual understanding of language using the Transformer architecture.

Model Architecture and Configuration

- **Pre-trained Model:** `bert-base-uncased` from the HuggingFace Transformers library.
- **Tokenizer:** BERT WordPiece tokenizer was used to convert input text into token IDs, with special tokens `[CLS]` and `[SEP]` added.
- **Classifier Head:** We used the `BertForSequenceClassification` class which adds a dropout layer and a linear classification layer on top of the BERT encoder.
- **Number of Labels:** The model was configured for multi-class classification with N output labels (where N is the number of distinct categories in the dataset).

Data Preprocessing

- Texts were lowercased and tokenized using the BERT tokenizer with a `max_length` of 128 tokens.
- Sequences were padded or truncated to ensure uniform input size.
- Attention masks were generated to differentiate between real tokens and padding tokens.
- The dataset was split into training and validation sets using an 80/20 split.

Training Details

- **Optimizer:** AdamW optimizer with weight decay fix.
- **Learning Rate:** 2×10^{-5}.
- **Batch Size:** 16.
- **Epochs:** 4.
- **Loss Function:** Cross-entropy loss.
- **Scheduler:** Linear learning rate scheduler with warm-up.

Training was performed using a GPU-enabled environment to accelerate the computation. Gradients were clipped to avoid exploding gradients.

Evaluation and Results

- **Training Accuracy:** Achieved over 98% accuracy on the training dataset by the final epoch.
- **Validation Accuracy:** Approximately 96% on the validation set.
- **F1 Score:** The weighted F1-score was recorded at 0.96, demonstrating high performance across all classes.
- **Loss Trend:** Both training and validation loss showed a consistent downward trend, indicating successful learning without overfitting.

Conclusion

The BERT-based classifier significantly outperformed traditional machine learning approaches, benefiting from its deep contextual representation of text. The model's fine-tuning process was stable and yielded high performance on both training and unseen data, confirming the suitability of transformer-based models for complex NLP classification tasks.