Twitter Sentiment Analysis

Yu Xie yx3774@nyu.edu
Carrie Song carrie.song@nyu.edu
Hang Liu hl4132@nyu.edu
Ziheng Feng zf2345@nyu.edu
Michelle Tong mwt7170@nyu.edu

1 Introduction

This project tackles the task of Twitter sentiment classification by incorporating not only textual content but also user behavior and temporal context to improve predictive accuracy. Using the dataset found on Kaggle, we build a reproducible preprocessing pipeline that cleans raw tweets and extracts a diverse set of features, including text, Word2Vec embeddings, temporal statistics, and user-level metadata.

We experiment with several classification models — Support Vector Machine(SVM), Random Forest, and XGBoost — trained on a standardized and PCA-reduced feature set. To ensure realistic evaluation, we design a cross-validation strategy that preserves temporal order (TimeSeriesSplit) and prevents user-based data leakage through grouped splits.

Finally, we benchmark each model using precision, recall, F1-score, and accuracy, while optimizing runtime efficiency with parallelized preprocessing and validation using Joblib. Our goal is to deliver a sentiment classifier that balances interpretability, scalability, and robust performance in dynamic social media settings.

2 Pre-processing

The analysis pipeline began with data preprocessing to transform raw Twitter data into a suitable format for machine learning models. This process involved several key steps to clean the text and extract informative features.

2.1 Data Preparation

The dataset contains tweets labeled as 0 (negative) or 1 (positive), along with user, timestamp, and other related metadata. Preprocessing included converting dates to datetime format and extracting fundamental metadata features. After that, we performed text cleaning — lowercasing text, converting hashtags to plain words, removing URLs, punctuation, special characters, and @mentions, and normalizing whitespace.

2.2 Feature Engineering

We extracted various features, including text-based, temporal-based, user-based, and semantic. To reduce dimensions, Principal Component Analysis (PCA) was applied to numerical features, preserving around 75% variance with the top 4 components.
The final feature preparation involved standardizing numerical features from PCA, one-hot encoding categorical features such as hour, day of the week, and month, and combining them with Word2Vec embeddings.

3 Model

We trained several classification models to predict the sentiment of tweets based on the features we engineered from the dataset, which were mentioned in the previous section. Then we compared different models and determined which performs best.

3.1 SVM

Overview
SVM serves as our baseline model due to its effectiveness in handling high-dimensional feature spaces and ability to find optimal decision boundaries in complex feature spaces
Implementation
For SVM, we used scikit-learn's SVC with an RBF kernel, configured with hyperparameters C = 10, gamma = 'auto', probability = False for faster prediction since probabilities are not needed.

3.2 Random Forest

Overview
Random Forest was selected for its ability to handle non-linear relationships and built-in feature importance evaluation, providing both performance and interpretability.
Implementation
The Random Forest classifier utilized an ensemble of decision trees implemented using scikit-learn's RandomForestClassifier with bootstrap sampling, configured with 200 trees and a maximum depth of 20.

3.3 XGBoost

Overview
XGBoost was employed for its gradient boosting framework, which sequentially corrects errors and delivers strong performance on classification tasks.
Implementation
The Gradient Boosting classifier (XGBoost) was implemented using scikit-learn's GradientBoostingClassifier, configured with 200 estimators with a learning rate of 0.1 and maximum depth of 3.

3.4 Ensemble Model

Overview
An ensemble voting classifier combining all three models to potentially achieve higher accuracy through model complementarity.
Implementation
Combines SVM, Random Forest, and XGBoost using majority voting. An ensemble model is expected to balance out the individual weaknesses of the base models.

3.5 Model Comparison

Accuracy is the primary metric we use to assess model performance, measuring the correctly classified tweets. The SVM model demonstrated the best overall performance, followed by the Ensemble model, XGBoost, and Random Forest. The relatively close performance metrics suggest that all models captured similar patterns in the data, indicating balanced performance rather than bias toward a particular sentiment.

Model	Accuracy	Precision	Recall	F1-score
SVM	0.7451	0.75	0.74	0.74
Random Forest	0.7306	0.73	0.73	0.73
XGBoost	0.7363	0.74	0.74	0.74
Ensemble	0.7413	0.74	0.74	0.74

Table 1: Comparison of model performance on the same test set.

4 Optimization

4.1 Python Code Optimization

In order to make our Python codebase run faster, we started with what we learned in lectures by focusing on optimizing our loops and functions. One of the first things we tackled was pulling repeated calculations out of loops. It cleaned up a lot of redundant processing in our iterative tasks. For example, when we were extracting text features, we initially found ourselves applying functions row by row. We switched that up by using `pandas` Series methods like `.str.len()` and `.str.count()`. These work on whole columns at once and proved way faster than the old for-loops or `.apply()` methods. Another key change was how we handled data structures. In functions like `extract_word2vec_features`, we started pre-allocating NumPy arrays (e.g., `vec = np.zeros(vector_size)`) to their full size right at the start. This stopped our data structures from constantly resizing inside loops, which made execution quicker and more predictable, especially when we threw large datasets at it.

4.2 Parallel Processing Optimization

Beyond those targeted fixes, we got a serious performance boost by bringing in parallel computing across different parts of our pipeline.Take data loading and preprocessing. We implemented a parallel approach using the `multiprocessing` module. The idea was to divide the dataset into chunks and have multiple CPU cores chew through them at the same time. Our tests showed this was a solid boost: our `load_and_preprocess_data_parallel` function, for instance, cut data ingestion and initial transformation time from about 23.1 seconds down to just 10.3 seconds.

We applied a similar strategy to our cross-validation. Using `joblib.Parallel`, we could evaluate different folds for both our time-based and user-based cross-validation concurrently. This presented a major improvement: time-based cross-validation went from 53.7 seconds to 25.5 seconds, and user-based cross-validation dropped dramatically from 87.8 seconds to 25.2 seconds.

Finally, we also revamped our `train_evaluate_models` function so it could train our machine learning models (SVM, Random Forest, XGBoost) in parallel. By leaning on `joblib.Parallel` again, and setting up models like Random Forest to use all available cores (`n_jobs=-1`), we managed to bring down the total model training time from roughly 588 seconds to 408 seconds. All these parallelization steps combined made our whole analytical workflow much snappier and better equipped to handle bigger tasks.

Task	Initial Runtime (sec)	Optimized Runtime (sec)	Speedup (%)
Dataload	23.00	11.00	52%
train_evaluate_models()	587.86	408.22	30%
Cross Validation	87.79	25.15	71%

Table 2: Performance Metrics Post-Optimization

5 Conclusion

In conclusion, our pipeline delivered robust predictive performance on the test set. SVM achieved 74.51% accuracy, Random Forest 73.06%, and XGBoost 73.63%; a hard-voting ensemble further raised accuracy to 74.13% with an F1 score of 0.74. PCA reduced the feature space to just four components while retaining 70.5% of the original variance, confirming the value of targeted dimensionality reduction.

On the optimization front, we vectorized Pandas string methods and pre-allocated NumPy arrays to eliminate Python loop overhead. Multiprocessing for data ingestion cut load/preprocessing time from 23.0 s to 11.0 s (52% faster). Parallel cross-validation via joblib reduced CV time from 87.8 s to 25.2 s (71% faster). Finally, by setting `n_jobs=-1` for the Random Forest estimator and dispatching the SVM and Gradient Boosting folds in parallel with `joblib.Parallel`, we cut total training time from 587.9 s to 408.2 s—an overall speed-up of roughly 30%.

Overall, combining rich, heterogeneous features with PCA and systematic parallelism yields a scalable, high-throughput pipeline that can keep pace with real-time social-media streams while maintaining state-of-the-art accuracy.

6 Github Link

https://github.com/m1tong/DSGA1019.git