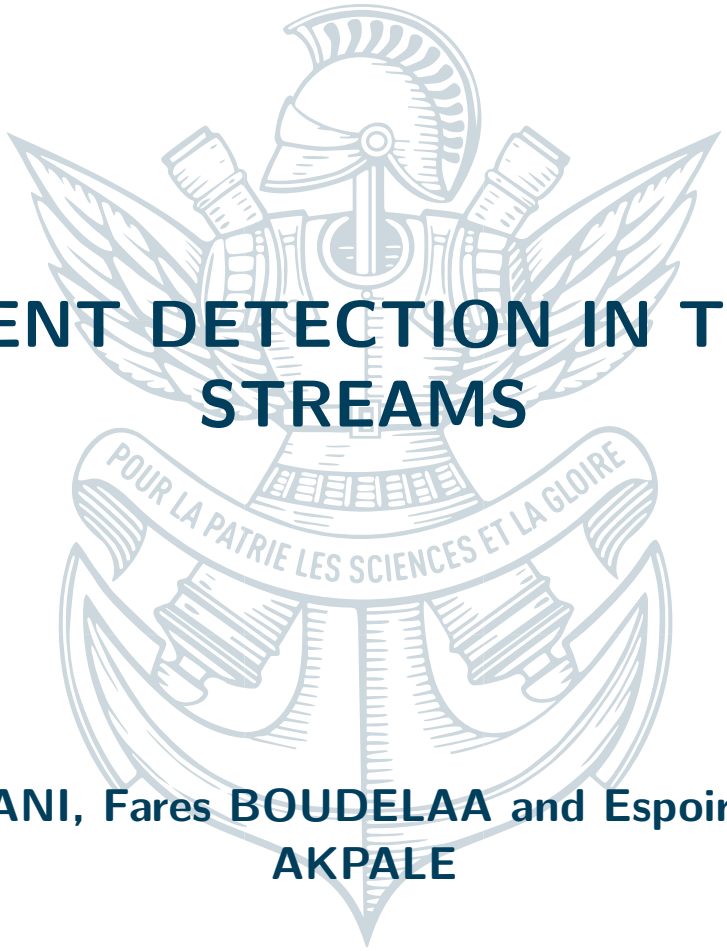

SUB-EVENT DETECTION IN TWITTER STREAMS

Gloire LINVANI, Fares BOUDELAA and Espoirs Emmanuel AKPALE

April 5, 2025

Kaggle Team name: 3 Loss Minimizers

Abstract

In this work, we address the challenge of detecting notable events within one-minute intervals of football matches using Twitter data from the 2010 and 2014 FIFA World Cup tournaments provided by [MXNV18]. We present a comprehensive approach that includes data preprocessing, feature engineering, and model development. Our solution achieves a significant improvement over the baseline models through careful feature selection and model selection.

Contents

1 Introduction

The task of sub-event detection in social media streams presents unique challenges in natural language processing and machine learning. Given Twitter data from football matches organized in one-minute intervals, our goal is to accurately classify whether a significant event occurred during each interval. This binary classification problem requires careful consideration of text preprocessing, feature selection, and model architecture to capture the temporal and contextual nature of match events.

2 Data Preprocessing and Feature Engineering/Selection

Our preprocessing pipeline involves several fundamental stages designed to improve the quality and informativeness of Twitter data. We have adopted a comprehensive framework that integrates traditional text preprocessing techniques with some of the latest methods in natural language processing.

2.1 Text Cleaning and Normalization

We first apply several cleaning operations to reduce noise while preserving contextually relevant information:

- Removal of URLs, and irrelevant special characters, which typically don't contribute to event detection
- Preservation of relevant punctuation (!, ?, :, ., ;) for sentiment analysis and BERT embeddings to preserve the natural structure of the sentence, an to leverage the fact that they often indicate emotional intensity associated with significant events
- Retention of score patterns (e.g., "1-0", "2:1") in tweets which are crucial indicators of goal events
- Suppression of duplicate tweets and retweets as in [MXNV18] to reduce noise and the number of tweets processed
- For GloVe embeddings, we tokenized the tweets using Python NLTK's TweetTokenizer.

Notably, we maintain stopwords that are relevant to football events (e.g., 'score', 'penalty', 'goal', 'card') while removing others. This selective approach preserves domain-specific information that might be crucial for event detection.

2.2 Feature Engineering

We engineered several features based on linguistic and temporal characteristics of tweets:

2.2.1 Temporal Features

We introduced the concept of "key periods" in matches, defined as moments where significant events are more likely to occur:

- Match kickoff (0 minutes)
- Half-time (45 minutes $\pm$ margin)
- Full-time (90 minutes $\pm$ margin)
- Extra time periods

We implemented a margin of ± 5 minutes around these periods to account for slight variations in the timing reports.

2.2.2 Linguistic Features

We extract several linguistic markers that could indicate significant events:

- Score mention detection using regular expressions
- Repeated character patterns (e.g., "GOOOAL")
- Exclamation and question mark counts
- Ratio of uppercase characters (often indicating excitement)

2.2.3 Sentiment Analysis

We employ a fine-tuned version [Har22] of DistilRoBERTa base model to extract emotional features from tweets. This model classifies emotions into seven categories: anger, disgust, fear, joy, neutral, sadness and surprise. The emotional intensity often correlates with significant match events, particularly joy, surprise, and anger during goals or dramatic moments.

2.3 Text Embeddings

2.3.1 BERT Embeddings

We utilize BERT (Bidirectional Encoder Representations from Transformers) [DCLT19] for our primary text representation. BERT's contextual embeddings capture rich semantic information through:

- Bidirectional context consideration
- Pre-training on a massive corpus of text
- Attention mechanism that captures relationships between words

We use the 'bert-base-uncased' model and implement batch processing for efficiency. Each tweet is encoded into a 768-dimensional vector by averaging the last hidden states across all tokens.

2.3.2 GloVe Embeddings

While we initially experimented with GloVe (Global Vectors for Word Representation) [PSM14] using the Twitter-trained 200-dimensional vectors, our empirical testing showed that BERT embeddings consistently outperformed GloVe for our specific task. This aligns with recent research showing the superiority of contextual embeddings over static word embeddings for event detection tasks. Consequently, our final models exclusively use BERT embeddings.

2.4 Feature Selection Strategy

Our feature selection process was guided by domain knowledge (football) and empirical testing. We maintained features that showed significant correlation with event detection in our validation tests. Particularly, the combination of BERT embeddings with our engineered features provided complementary information: while BERT captures semantic content, our custom features capture domain-specific patterns and temporal context that might not be apparent in the raw text alone.

3 Prediction

3.1 Tweet Aggregation and Feature Preparation

Before model training, we needed to aggregate multiple tweets from each one-minute interval into a single feature vector. We implemented several aggregation methods to capture the temporal and semantic relationships between tweets.

3.1.1 Temporal Feature Aggregation

For non-embedding features, we compute both simple statistics and temporal signals:

- Mean aggregation for sentiment features
- Sum aggregation for count-based features (exclamation marks, question marks)
- Tweet count per interval as a measure of period activity
- Lagged features to capture temporal dynamics:
  - First-order differences ($x_t - x_{t-1}$) for certain non embedding features and squared L_2 Norm of one-step differences between embedding vectors to model topic change
  - One-step lag values (x_{t-1}) to model temporal dependencies

3.1.2 Embedding Aggregation Methods

In addition to simple mean aggregation of BERT embedding vectors, We experimented with two other approaches:

1. Attention-based Aggregation Following the principles of self-attention [VSP+17], given a matrix of tweet embeddings E, we compute:

$$\text{Attention}(E) = \text{softmax}\left(\frac{EE^T}{\sqrt{d}}\right)E \tag{1}$$

where E is our matrix of tweet embeddings and d is the embedding dimension. The softmax operation creates attention weights that determine how much each tweet contributes to the final aggregated representation.

2. Similarity-based Aggregation We compute cosine similarities between tweet embeddings to weight their contributions:

$$w_i = \text{softmax}\left(\frac{1}{N}\sum_{j=1}^{N}\frac{e_i^T e_j}{\|e_i\|\|e_j\|}\right) \tag{2}$$

where e_i and e_j are tweet embeddings, and N is the number of tweets in the period.

Mean aggregation and attention-baed aggregation performed similarly, while cosine similarity based aggreation drastically reduced models performance. This can be explained by the fact that this similarity measure is vulnerable to noise, as explained in [vdBP19]

3.1.3 Dimensionality Reduction and Normalization

To manage the high dimensionality of our feature space while preserving important signals, we:

- Apply PCA to reduce feature vectors to an optimal number of 46 components, capturing over 95% of the variance (we compared a fairly wide range of PCA components)
- Standardize all numerical features using Z-Score normalization

3.2 Model Selection and Optimization

3.2.1 Baseline Models

We evaluated four different models as our baselines, each chosen for their specific strengths in binary classification tasks:

- **Logistic Regression**: A linear model that serves as our primary baseline due to its effectiveness with high-dimensional data and because it's a reference model for binary classification tasks
- **Random Forest**: An ensemble method that can capture non-linear relationships and handle feature interactions
- **LightGBM**: A gradient boosting framework known for its efficiency with large datasets
- **XGBoost**: Another gradient boosting method that often performs well on tabular data

In our initial evaluation without hyperparameter optimization, Logistic Regression emerged as our best performing model, achieving approximately 0.79 accuracy on the training set and 0.75 on the validation set. This relatively small gap between training and validation performance indicates good generalization. This base model is also our final model, which achieved the highest accuracy on the test set.

Interestingly, while tree-based models (Random Forest, LightGBM, and XGBoost) achieved higher training accuracies around 0.96, they showed signs of overfitting with validation accuracies around 0.78. This overfitting behavior suggests that despite their theoretical advantages in capturing non-linear relationships, the tree-based models might be learning noise patterns in our high-dimensional feature space.

3.2.2 Hyperparameter Optimization

To improve model performance, we conducted extensive hyperparameter optimization using GridSearchCV with 5-fold cross-validation. Our parameter grids were designed based on the following considerations:

Logistic Regression We explored multiple solver-penalty combinations:

- SAGA solver with $L1$, $L2$, and elasticnet penalties to handle different regularization needs
- LBFGS solver with $L2$ penalty for better convergence on smaller datasets
- LIBLINEAR solver for efficiency with $L1$ regularization
- C values ranging from 10^{-4} to 10^{4} to control regularization strength

Tree-based Models For Random Forest, LightGBM, and XGBoost, we focused on parameters that control model complexity and prevent overfitting:

- **Tree Structure Parameters**:
  - max_depth: Limited to shallow trees (3-5 levels) to prevent overfitting
  - min_samples_split/min_child_samples: Higher values (10-30) to ensure robust splits
  - subsample/bagging_fraction: Values around 0.6-0.8 for better generalization
- **Learning Parameters**:
  - learning_rate: Small values (0.01) for more stable learning
  - n_estimators: Moderate values (50-100) balanced with early stopping
  - reg_alpha/reg_lambda: $L1$ and $L2$ regularization to control model complexity
- **Class Imbalance Handling**:
  - class_weight: Explored both balanced and custom weights based on class distribution
  - scale_pos_weight: Adjusted for class imbalance in XGBoost

We also incorporated early stopping during the training process to prevent overfitting, with a patience of 50 epochs for gradient boosting models. The parameter grids were intentionally kept moderate in size to balance between exploration of the parameter space and computational efficiency.

3.2.3 Neural Network Model

We also evaluated a simple Multi-Layer Perceptron, composed of three fully dense Layers, a batch normalization layer and an activation layer as seen in Table 1.

Layer (type)	Output Shape	Param #
Batch Normalization	(None, 781)	3,124
Dense	(None, 700)	547,400
Dropout	(None, 700)	0
Dense (Dense_1)	(None, 200)	140,200
Dropout (Dropout_1)	(None, 200)	0
Dense (Dense_2)	(None, 120)	24,120
Dense (Dense_3)	(None, 1)	121

Table 1: Summary of Neural Network Layers

- **Model Architecture**:
  - Batch Normalization Layer: The model begins with a batch normalization layer. Since the data has high variance (e.g., names of countries and players appearing in only a few tweets), normalizing the data helps mitigate this. Additionally, batch normalization increases the convergence speed of the model, allowing the use of a higher learning rate.
  - Fully Connected Layer: The model applies 3 fully connected neural network layers. The output dimensions of these layer were determined through experimentation. While an output size of 1500 performed better on the validation set, an output size of 700 achieved better generalization on new data, *ceteris paribus*.
  - Dropout Layer: To reduce overfitting, a dropout rate of 60% was used. This rate was found to strike the ideal balance between promoting learning and preventing overfitting.
- **Hyperparameters**:
  - Batch Size: A batch size of 64 was chosen as it balances the noise in training, which acts as a shield against overfitting, while retaining sufficient information for generalization.
  - Regularization: L_2 regularization was applied to each layer with a regularization rate of 0.001. This penalizes high weights while preserving the advantages of a high learning rate.
  - Early Stopping: Early stopping was employed with a patience of 25 epochs and a minimum delta of 0.005. This served as a safeguard against overfitting.
  - Optimization Algorithm : The ADAM optimizer (Adaptive Moment Estimation) was used to iteratively update weights. Paired with a high learning rate of 0.01, it performed well, thanks to the inclusion of batch normalization.
  - Activation Function: The ReLU activation function was applied. Although the tanh activation function increased the F1 score by 2-3 points on the validation set, ReLU produced better results on the test set in the final challenge.
- **Interpretation and Insights**

  The MLP achieved better performance than baseline logistic regression; however, it failed to outperform other models when tuned with hyperparameters.

  We believe this is due to its limited ability to generalize to words never seen before. A more complex model, such as a Convolutional Neural Network, would likely have performed better by capturing patterns within the tweets.

  An ensemble approach could also have been effective, such as stacking the MLP with XGBoost for improved predictions. Nevertheless, the MLP provides valuable insights into the data and serves as a strong baseline for more advanced approaches.

4 Conclusion

In this project, we presented an effective approach for sub-event detection in Twitter streams during football matches. Our preprocessing pipeline and feature engineering strategies, combined with a Logistic Regression although simple, demonstrate significant improvements over baseline models. Future work could explore incorporating Large Language Models for more sophisticated semantic understanding, or graph-based approaches to model temporal and semantic relationships between tweets. These methods could potentially capture more complex patterns in social media streams and better understand the contextual nature of football match events.

References

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[Har22] Jochen Hartmann. Emotion english distilroberta-base. `https://huggingface.co/j-hartmann/emotion-english-distilroberta-base/`, 2022.

[MXNV18] Polykarpos Meladianos, Christos Xypolopoulos, Giannis Nikolentzos, and Michalis Vazirgiannis. An optimization approach for sub-event detection and summarization in twitter. In *European Conference on Information Retrieval*, 2018.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[vdBP19] Tim vor der Brück and Marc Pouly. Text similarity estimation based on word embeddings and matrix norms for targeted marketing. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1827–1836, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.