KAGGLE REPORT
INF 554

September 24, 2025

Sara El Baghdadi | sara.el-baghdadi@polytechnique.edu
Mathias Perez | mathias.perez@polytechnique.edu
Hippolyte Wallaert | hippolyte.wallaert@polytechnique.edu

1 Data Preprocessing and Feature Selection/Extraction

Cleaning and preprocessing the data: Our approach involved the following key steps:

Cleaning the data:

- **Removing duplicates:** We removed duplicate tweets as irrelevant spam, reducing the dataset size by half.
- **Filtering out empty tweets:** We removed tweets that were empty or contained only whitespace.
- **Removing URLs:** Tweets containing URLs were excluded, as they are often considered noise.
- **Removing RT or @ tweets:** Such tweets directly address another user, providing no match information or delayed updates.

Preprocessing the data before embedding:

- **Removing Hashtags:** We removed hashtags as they provide no meaningful information.
- **Handling Emojis:** Emojis and non-ASCII characters were removed by encoding tweets to ASCII and decoding them back.
- **Tokenization - Removing stop-words - Lemmatization:** We performed standard text preprocessing tasks using **nltk**[1].

Embedding:

Embedding per tweet:

After cleaning the data, we quickly moved on to encoding the tweets. We experimented with **GloVe**[5], **Google's Universal Sentence Encoder**[2], and **BERT-based** embeddings[3] for tweet representation.

First, we implemented the GloVe embedding method used in the baseline model, generating fixed-size 200-dimensional embeddings for each tweet. However, we hypothesized that directly generating embeddings for entire tweets, rather than averaging word embeddings, could improve performance. This led us to test full-sentence embeddings using Universal Sentence Encoder and BERT.

For BERT-based embeddings, we used the lightweight **DistilBERT**[6] model, which generates 768-dimensional embeddings:

- The tokenizer and model were loaded from the pre-trained 'distilbert-base-uncased' model.
- Tweets were tokenized and passed through the model to extract hidden states from the last layer.
- Mean pooling was applied to generate a single 768-dimensional embedding per tweet.

These embeddings were added to the dataset, where each tweet is now represented by a 768-dimensional vector (DistilBERT), a 512-dimensional vector (Universal Sentence Encoder), or a 200-dimensional vector (GloVe). Ultimately, GloVe embeddings proved to be the most effective, as their lower dimensionality aligned better with the tweets' simplicity and the dataset's limited size (in terms of number of periods).

Embedding per period:

Once the embeddings were obtained for each tweet, we explored several ways of aggregating the tweets within a period.

We experimented with both mean and max pooling for the 512-dimensional tweet embeddings and trained a RandomForest model. A Shapley analysis highlighting the marginal contributions of individual features revealed that mean pooling was significantly more important than max pooling, leading us to choose mean pooling for period representation. Indeed mean pooling effectively captures the overall context of a period by averaging the semantic contributions of all tweets, whereas max pooling tends to prioritize outlier tweets, which may not represent the broader trends within the period. We also attempted to cluster tweets within periods to represent each period by its centroids, but the data lacked a clear cluster structure, so we abandoned the approach.

Given that we ended up with more than 200 embeddings per period, we considered the possibility that this dimensionality might be too large. To address this, we explored applying Principal Component Analysis (PCA)[7] to reduce the embeddings' dimensionality. For example, with DistilBERT embeddings, 90% of the variance is captured by just 35 components out of the original 768, indicating that PCA could effectively compress the data without significant information loss.

Feature Engineering: The development of our features was driven by strong intuitions, each of which was systematically validated during experimentation.

- **Tweet count and volume growth:** Our analysis confirmed that the growth in tweet volume per period is a highly discriminative feature for detecting events.

  Figure 1 and Appendix 1 demonstrate this insight, showing clear peaks in tweet activity during significant events in at least 11 out of 19 matches. These peaks align even more distinctly with periods labeled as EventType 1. Moreover, as shown in Appendix 2, we identified over 150 periods in the training dataset as EventType 1 with 90% certainty based on increased tweet volume compared to the previous period.

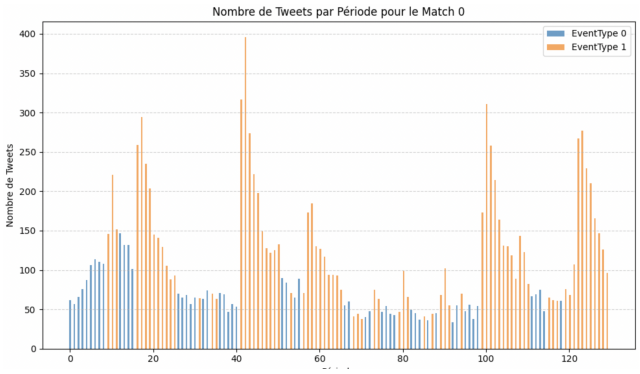


Figure 1 Tweet activity peaks linked to EventType 1 in Match 0 periods.

- **Presence of specific words:** We hypothesized that certain words such as "goal" or "card" would be commonly

mentioned during significant events. To validate this, we performed a TF-IDF analysis while excluding country names (cf. below). This study revealed several highly discriminative words, as per Figure 2.

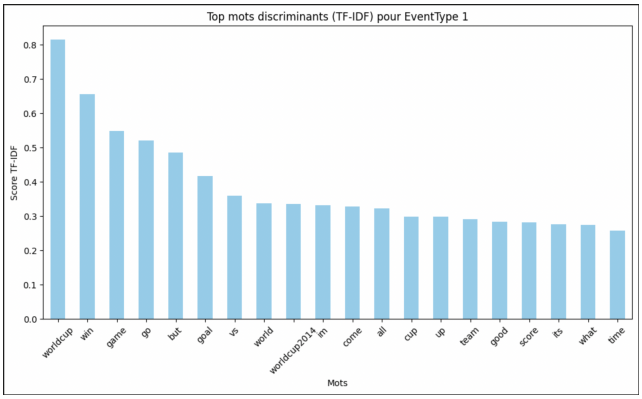


Figure 2 TF-IDF

We further assessed the significance of each word by analyzing its frequency and testing whether a threshold could reliably identify EventType 1 periods with at least 90% certainty. For example, as shown in Appendix 3, applying a frequency threshold of 17 for the word "goal" enabled the identification of over 50 EventType 1 periods with high confidence. From these results, we selected the following words as features: `['world', 'goal', 'halftime', 'fulltime', 'yellow', 'card', 'red', 'offside', 'foul', 'substitution', 'well played']`, with functionality to account for elongated forms (e.g., "goaaal").

- **Country placeholder:** In our analysis, country names like 'arg', 'germany', and 'brazil' proved highly discriminative. However, learning these words wouldn't generalize to matches involving other countries. To address this, we replaced all country names with the placeholder 'country'.
- **PeriodID:** We kept `PeriodID` because we observed that the last few periods, as well as the middle periods of a match, tend to have more significant events.
- **Tweet characteristics:** Another intuitive feature was the average number of exclamation marks, digits, and capital letters per tweet in each period. Metrics for EventType 1 and EventType 0 showed significant differences, and these features consistently ranked highly in Shapley Analysis. This further validated their inclusion in the final model.

Feature Selection Impact on predictive performance: Feature selection was crucial for improving model performance, reducing overfitting, and enhancing generalization, particularly given our limited dataset size. We used multiple algorithms to evaluate feature importance and reduce dimensionality.

Recursive Feature Elimination (RFE) with Random Forest and XGBoost effectively identified a minimal subset of features that maximized accuracy. We dropped word frequency features for `['penalty', 'let go', 'well done', 'come', 'go go go']` as well as emoji frequency.

As shown in Figure 3, Shapley Additive Explanations (SHAP) provided insights into feature contributions. Many features added through feature engineering appear in the top 20 out of 221 features. For example, the growth in the number of tweets from one period to the next, the standardized number of tweets per period, and the average number of exclamation marks per post in a period. This further supports the value of our feature engineering process.

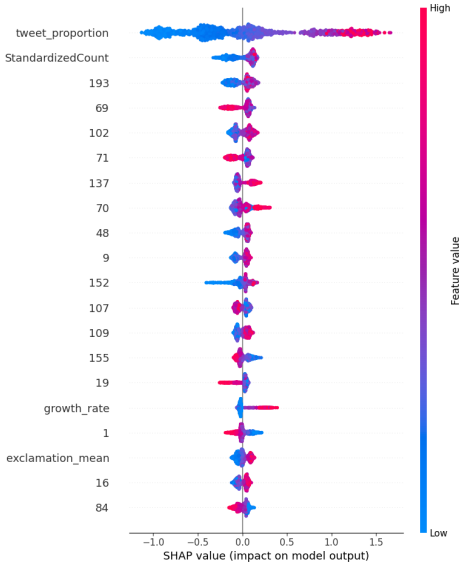


Figure 3 Top 20 most important features for our XGBoost model according to SHAP values

Feature selection reduced overfitting, improved XGBoost's validation accuracy and enhanced training efficiency by simplifying the input space, lowering computational costs, and enabling more thorough hyperparameter tuning.

2 Model Choice, Tuning and Comparison

'Non-temporal' methods: We explored several classical classifiers available in scikit-learn and Deep learning methods, tuning their hyperparameters to optimize performance. The models we tested include Logistic Regression, Support Vector Machine (SVM), Random Forest, Fully Connected Neural Network (FC), and a Deep Neural Network (DNN) with an attention layer. After optimizing the models, we compared their accuracies, as shown in the table below. **Our best model is a finetuned XGBOOST, with 74.609% on the available Kaggle eval set.**

Table 1 - Accuracy of different classifiers after hyperparameter tuning.

Model	**Accuracy***
Logistic Regression	68.2%
Support Vector Machine (SVM)	52.8%
Random Forest	79.3%
XGBOOST	**77.3%**
Fully Connected DNN	72.2%

* The accuracies are calculated on test samples created with *train_test_split*.

We finetuned our models using random search first, as it is quicker. Once we identified promising regions, we switched to grid search to refine the hyperparameters with more precision. Additionally, we paid attention to the effect of hyperparameters and prioritized those that led to less overfitting.

Discovery of Time Series Behavior: We observed a temporal pattern in the data: event type 1 often persisted across multiple periods in match 0 (see Figure 1). To investigate this, we calculated the correlation between consecutive periods, shown in Figure 4.

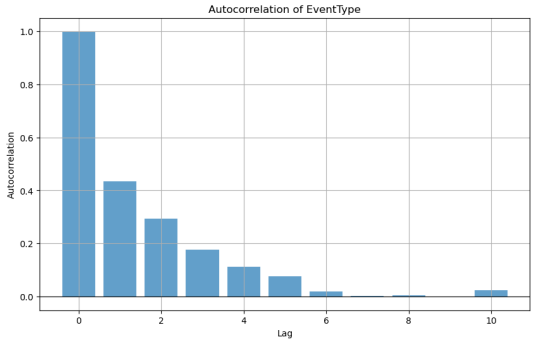


Figure 4 Autocorrelation of EventType for different lags.

To further analyze the relationship between consecutive periods, we constructed a transition matrix (see Appendix 4) showing the probabilities of transitioning between event types, reinforcing evidence of temporal dependencies.

Based on this insight, we incorporated context by defining time intervals centered around each period for our time series models. We then explored methods to maximize the value of this additional context.

'Temporal' methods: Observing that the evolution of certain features significantly influences model predictions, we decided to use Long Short-Term Memory (LSTM) models[4] to better capture the temporal dynamics within each period's context. Specifically, bidirectional LSTMs were employed to effectively account for variations in both past and future sequences.

However, a simple Bidirectional LSTM model alone was not enough to capture all the intricate sub-features from a sliding window. To address this limitation, we proposed **stacking a Deep Neural Network (DNN) with the Bidirectional LSTM** into a unified model that combines the strengths of both approaches. Its architecture is illustrated in Figure 5.

This combined model, being more complex, presented greater challenges during training. Striking the right balance between training capacity and overfitting was crucial to achieving optimal performance. Our best model achieved an accuracy of 78% on the training set and 76% on the validation set. However, when submitted to Kaggle, the performance dropped to 71%, likely due to overfitting or distribution differences in the test dataset.

Despite our efforts we were never able to surpass the Kaggle performances of our simpler XGBoost based models, probably due to the complexity of our model, prone to too much overfitting.

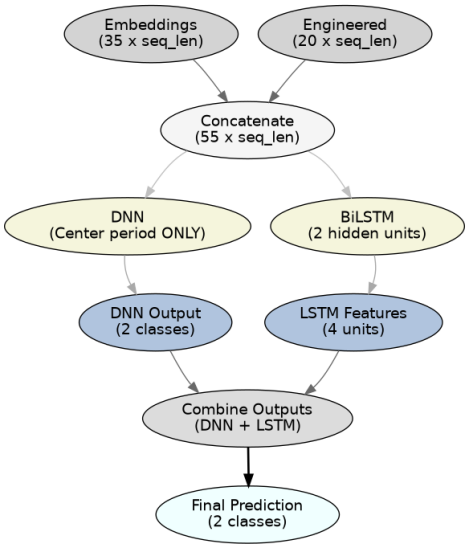


Figure 5 Stacked bidir LSTM and DNN model.

3 Discussion and outlook

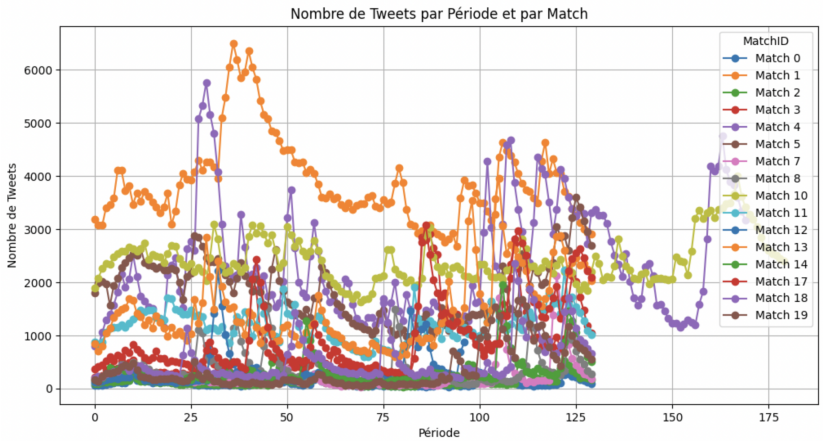
First of all, one of the main limitations we faced in this challenge was the difficulty in generalizing to the evaluation data. Even though the evaluation and training matches sometimes involved different teams, the extent to which our models unknowingly overfitted to specific games was surprising. We attempted to mitigate this using traditional data-splitting techniques or cross-validation methods, but the significant gap in accuracy between our local validation results and our Kaggle submissions persisted. It was only when we reserved entire matches exclusively for evaluation that we were able to reproduce this gap, confirming that **the issue was due to overfitting on match-specific patterns**.

Second, our study revealed limitations in applying complex temporal models like the Stacked LSTM. The primary challenge was the limited size of our dataset, with only 2137 periods, which was insufficient to fully train a deep learning model with a high number of parameters. This constraint led to overfitting, as the Stacked LSTM struggled to generalize effectively. **In contrast, simpler models, particularly XGBoost with well-engineered features, outperformed more complex approaches, highlighting the importance of adapting model complexity to the available data.**

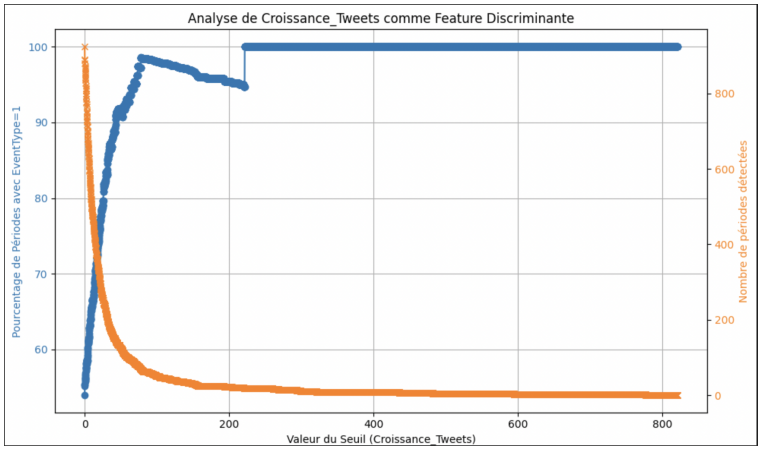
To address the issue of limited data, we could have explored **data augmentation** by splitting Period IDs into smaller segments and using these subsegments to train on a larger set of periods.

Possible future approach The next direction would be to fine-tune a large language model such as BERT, for direct event classification. Instead of relying on aggregated embeddings for each period, the LLM could process part of tweets in a period as a contextual input and classify the event type directly with a specific classification head. We guess that this approach along with relevant data augmentation will combine the advantages of our 'Temporal' and 'Non-temporal' methods, potentially yielding better results.

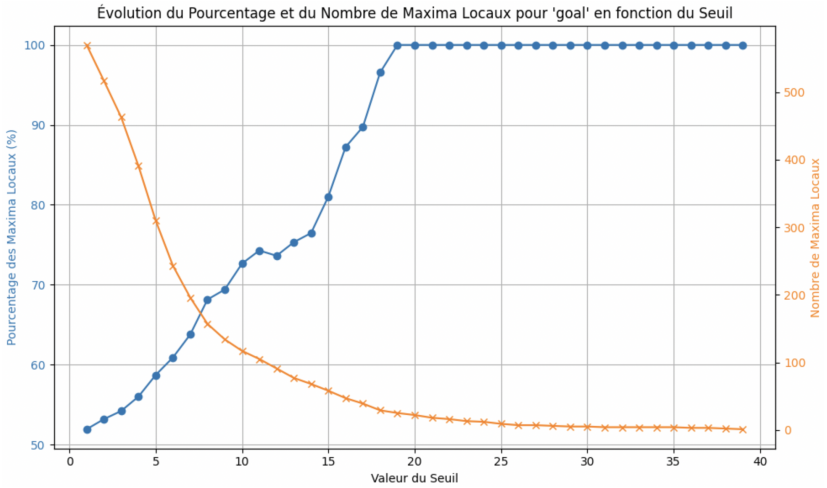
APPENDIX



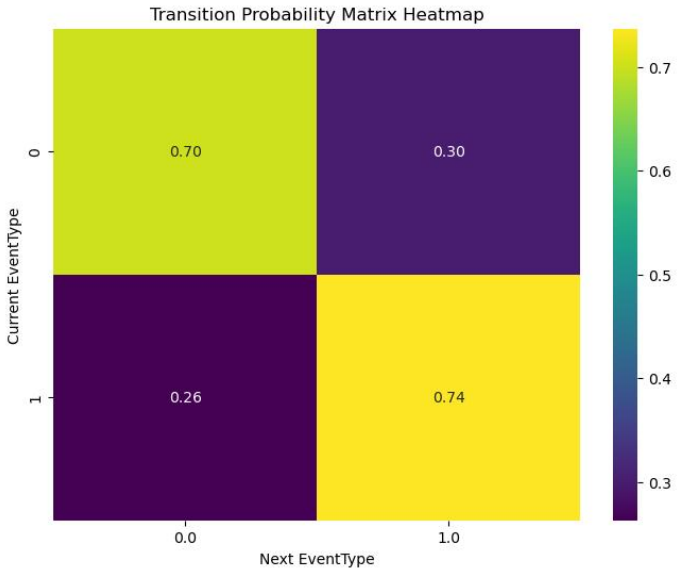
Appendix 1: Analysis of significant Tweet activity peaks across all matches.



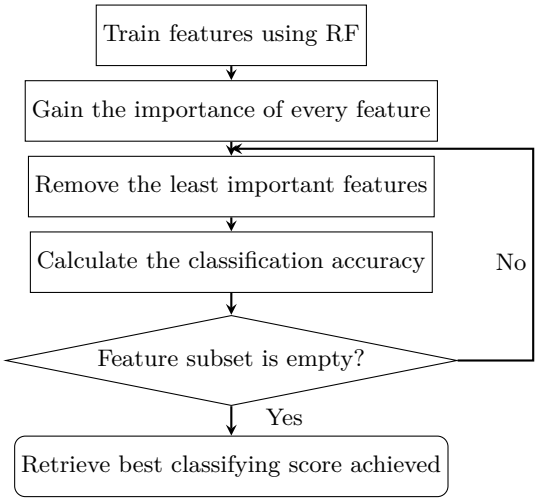
Appendix 2: Threshold analysis for detecting EventType '1' based Tweet volume growth



Appendix 3: Threshold analysis for detecting EventType '1' based on word frequency "goal"



Appendix 4: Transition matrix of EventType between consecutive periods.



Appendix 5: Flowchart of the recursive feature elimination (RFE) process.

References

[1] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, Inc., 2009. https://www.nltk.org/.

[2] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018. https://tfhub.dev/google/universal-sentence-encoder/4.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2019. https://github.com/google-research/bert.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[5] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. https://github.com/stanfordnlp/GloVe.

[6] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019. https://github.com/huggingface/transformers.

[7] Michalis Vazirgiannis. Machine and deep learning course material. https://moodle.polytechnique.fr/course/view.php?id=19308, 2024. Available on Moodle, École Polytechnique.