INF554 - KAGGLE Report

Victor DESHORS, Rodrigue REIBEL & Adrien BINDEL

1 Data Preprocessing and Feature

1.1 Cleaning and Preprocessing the data

We tried two models for embedding the tweets:

- **BERTweet**: A pre-trained bidirectional transformer model fine-tuned for tweets, effectively capturing contextual relationships in short, informal text.
- **GloVe**: A classical word embedding model generating fixed-dimensional representations based on word co-occurrence, requiring additional preprocessing for tokenization and out-of-vocabulary words.

1.1.1 Data Preprocessing for BERTweet

We focused on cleaning and standardizing the text data to enhance the model's performance. Since we use a pretrained model, we use the same preprocessing techniques for inference as those that were used for pretraining in BERTweet [1]. Preprocessing deals with abbreviations, link and mention management, consecutive letters (e.g. goooooaaaalll). Preprocessing functions can be found in *TweetNormalizer.py*. We also tried adding the following question at the beggining of each tweet, before embedding it : *"Does this tweet refer to a recent specific event in the football match ?"*. The intuition is that by adding context, the embeddings will change, and be more representative of the semantic content of the tweet that we care about. This intuition proved to be true, since this modification consistently improved the accuracy of the model. We hypothesize that these modifications reduce noise and standardize inputs, thereby enabling BERTweet to better capture the semantic content of tweets.

1.1.2 Data Preprocessing for GloVe

For the GloVe-based approach, we implemented additional preprocessing steps to clean the data and extract meaningful features:

- Removed non-semantic symbols (`'@'`, `'RT'`) and extracted two new features: mentions and retweets per minute.
- Replaced website links and replies with placeholders, in line with BERTweet preprocessing.
- Removed irrelevant smileys, emojis, and hashtags.
- Anonymized data by replacing country names with "team 1"/"team 2" and player names with "player from team 1"/"player from team 2".

1.2 Explanation of features

1.2.1 Tweet embeddings : clustering and subsampling

The main challenge of our classification task is to select meaningful features for each time period. The main features are the embeddings of the tweets, but they represent $N * d$ values per period, where N is the number of tweets and d the number of embedding dimensions. The baseline deals with this by averaging the embeddings over each time period. We tried two distinct methods :

- Clustering : we first apply an unsupervised clustering algorithm to separate different topics, and then average over the most populated one. The intuition behind this approach is that clustering can help identify distinct topics being discussed within a specific minute, allowing us to focus on the most prominent topic. We thus have one d-dimensional embedding per minute, and then use a binary classifier.
- Subsampling and CNN: we sample m tweet embeddings per minute, and classify the m-long sequence of embeddings using a 1-dimensional CNN. The idea is to recognize patterns in tweet embeddings at different time scales (from local to global).

1.2.2 Additional features

Since part of the information in the tweets is removed when averaging or subsampling, we decided to retain certain elements by creating the following features. The features include the number of tweets, mentions (`@`), consecutive repeated letters, exclamation marks, question marks, and smileys per minute. The rationale behind these features is that they may help identify peaks in activity, which could correspond to significant events. As every match is different and has different activities, we normalized each feature by the total sum during the match.

Given that the embedding dimensions are high (200 for GloVe and 768 for BERTweet), we applied dimensionality reduction using Principal Component Analysis (PCA). We experimented with the following approach : Applying PCA directly to the averaged embeddings for each minute after clustering. We chose PCA because it is a linear method well-suited for large datasets, allowing us to reduce dimensionality efficiently while retaining as much information as possible.

N_PCA	Mean Accuracy	Std Accuracy
10	0.5723	0.0458
25	0.5989	0.0688
50	0.6055	0.0622

Table 1: Accuracy moyenne et écart type pour différentes valeurs de N_PCA avec GloVe.

In the following, we use N = 50.

2 Model Choice, Tuning and Comparison

2.1 Unsupervised clustering

For clustering on the tweet embeddings, we tested two different models :

- KMeans : we used it as a baseline for its simplicity. There were several issues : it doesn't deal with noise, it works with euclidean distance, and the number of clusters is a hyperparameter.

- HDBSCAN: A density-based clustering algorithm that extends DBSCAN by analyzing the varying density of data points to automatically determine the optimal number of clusters and handle noise. This algorithm also allows for the use of cosine distance, which is more appropriate in BERT embedding space.

  We found HDBSCAN to be more performant, as we expected.

2.2 Supervised binary classification

We compared the models based on the average accuracy obtained from a 5-fold cross-validation. To ensure a robust evaluation, we decided to split our train/test data based on different minutes rather than different matches. We experimented with the following classifiers:

- **Random Forest:** Average accuracy was acceptable, but results were unstable, with some poor accuracy instances highlighting inconsistency.
- **Gradient Boosting (GradientBoostingClassifier):** Promising option due to its ability to incorporate penalty terms, with initial results showing potential for improved handling of data.
- **Logistic Regression:** Used as a benchmark, offering a simple model for comparing the performance of other classifiers.
- **XGBoost:** Chosen for its robustness, XGBoost handled noisy data well, with more consistent cross-validation accuracies centered around the mean.

After finding the best parameters doing a grid-search, we had the following performances on cross-validation :

Classifier	**Mean Accuracy (CV)**	**Std Accuracy (CV)**
XGBoost	0.5584	0.0941
Logistic Regression	0.6369	0.0529
Random Forest	0.5547	0.1103
Gradient Boosting	0.5995	0.1009
MLP	0.6416	0.0455
SVC	0.6392	0.0315
KNN	0.5910	0.0349

Table 2: Summary of classifier performances with cross-validation mean accuracy and standard deviation.

We will use SVC for classifying.

2.3 1-D Convolutional Neural Network

We also experimented with a 1-dimensional CNN binary classifier to avoid aggregating tweet embeddings for each time period. We first do a subsampling of size m on the N tweet embeddings for each period, and then apply our model on the sequence. We experimented with the number of layers, the sizes of convolution and pooling kernels and the parameters of the fully connected classification layer. We also added a pretraining phase consisting of next tweet prediction to prevent overfitting in the convolutional layers. Despite that and using randomization for data augmentation, we did not manage to prevent overfitting due to the low size of the dataset (number of periodIDs).

References

[1] Thanh Vu Dat Quoc Nguyen and Anh Tuan Nguyen. Bertweet: A pre-trained language model for english tweets. in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 9-14:2092–2115, 2020.