Machine and Deep learning Kaggle

Sub-Event detection in Twitter datastreams

DESHORS Victor, REIBEL Rodrigue, BINDEL Adrien

Section outline

Data Overview

- Number of tweets analyzed: 5,056,050.
- Total number of matches in the dataset: 16.
- Number of minutes : 2137.
- Dataset size: **large-scale**, with approximately 2,300 tweets per minute.
- **Labeling method**: Labels are assigned **collectively** to all tweets within a given minute.
- Main limitation: Too few matches to train models robustly, especially due to the heterogeneity in match characteristics.

Preprocessing Steps

- **Comparison of GloVe vs BERT:**
  - *Outputs:* GloVe generates static embeddings, whereas BERT provides contextualized embeddings.
  - Using *baseline* on BERTweet instead GloVe improved the accuracy of 4%.
- Common preprocessing for both methods: Removal of uninterpretable characters such as `@` (mentions) and `RT` (retweets).
- Enhanced BERT embeddings: Contextualization further improved by adding an additional sentence to clarify the context.
- Anonymization: Replaced country names with *Team 1* / *Team 2*, and player names with *Player from Team 1* / *Player from Team 2*.
- Differences in preprocessing requirements: BERT typically requires less preprocessing but may misinterpret certain messages.

New Features and PCA

- Created features: tweets count, mentions (@), repeated letters, exclamation/question marks, smileys per minute.
- Normalized each feature by the total sum during the match.
- Goal: Identify activity peaks potentially tied to significant events.
- Embedding dimensions (200 for GloVe, 768 for BERTweet) reduced using PCA.
- Applied PCA to averaged embeddings after clustering.
- PCA chosen for efficient dimensionality reduction on large datasets.

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

Table: Mean accuracy and standard deviation for different N_PCA values with GloVe and *baseline*.

Selected value: N = 50

Section outline

Clustering

Goal: Find a way to agregate the embeddings that represents well the content of the tweets for each minute

- isolating several clusters enables to identify the different topics of conversations
- useful for denoising and to identify the occuring or not of an event
- mise en oeuvre -> pseudo code

Algorithm 1 Embedding of the tweets with a clustering method

```
for minute in match do
    for tweet in minute do
        Compute embedding of tweet (GloVe or BERT)
    Get the clusters
    Keep only the embedding of the center of the main cluster
```

Choice of the clustering method

Algorithms tested: K-means and HDBSCAN

- K-means: only convex clusters
- K-means: fixed number of clusters, tests with k=2,3,4

→ HDBSCAN provides better results

Once we get the clusters, how to choose the final representation of the minute ?

- Tests with the mean of the main or the two main clusters
- We keep only the mean of the **mean of the main cluster**

Classifiers tested

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

Section outline

Method 2 : using a 1D CNN

Algorithm 2 Sequence classification with BERT + 1D CNN

for *tweet* in *tweets* **do**
  Compute BERT embedding of *tweet*
for *time_period* in *time_periods* **do**
  Sample N_{tweets} tweet embeddings
  Apply a 1D CNN binary classifier on input of shape $(768, N_{tweets})$

Motivation :

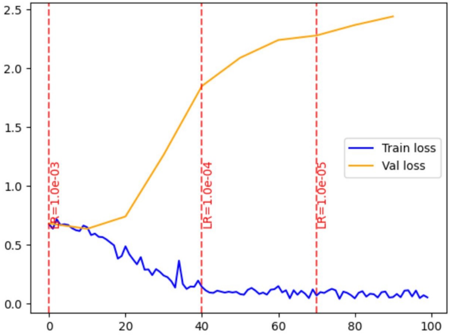
- Avoid aggregating tweet embeddings with an average
- Detect local and global patterns in tweet streams

Parameters of the model

- N_{tweets} : size of the sequence
- Convolutional layers :
  - number of layers
  - number of channels per layer
  - convolution kernel sizes / stride
- Final fully connected layer
  - Number and size of layers
  - activation function
- Training parameters
  - learning rate
  - regularisation

Main challenge : overfitting

- Small dataset ($\approx 1.5k$ time periods for training)
- High dimension of the input
- Leads to overfitting :



Some solutions

- Applying randomized subsampling for training
- Adding a pretraining phase :
  - Task = prediction of the average of the next K tweets
- Early stopping
- Use of smaller models

In the end, we managed to get promising results but very sensitive to the test set.

Conclusion

Public Kaggle score: 0.70703
Private Kaggle score: 0.67307

Technical Extensions:

- Perform cluster detection within clusters.
- Perform clustering per minute based on timestamp.
- Idea: Generate new matches randomly to increase the volume of data?