REPORT

Sub-Event Detection in Twitter Streams

Course: CSC_51054_EP - Machine and Deep learning

Authors: Vlada Voronina, Lauren Ramanantsoa
Emails: vlada.voronina@polytechnique.edu, lauren.ramanantsoa@polytechnique.edu
Team: tLV

Date: December 2024

Table of Contents

1 Introduction

This report presents our approach for detecting notable sub-events in football games based on Twitter streams. Our dataset consists of 5 million labeled tweets covering 16 football matches from 2010 and 2014. Each tweet includes an ID, match ID, period ID, event label, and timestamp, categorized into 1-minute time bins. The task is to predict whether an event occurred or not for each bin in an unlabeled test set of 4 new matches, aiming for maximum accuracy. The dataset is noisy, so we emphasize the importance of preprocessing steps. We apply both classical machine learning and deep learning methods to capture the linguistic patterns at both tweet and bin levels. In the end, we propose a Multi-Layer Perceptron (MLP) model that achieves a test accuracy of 0.738 for bin-level event classification.

2 Data Preprocessing and Feature Selection/Extraction

2.1 Cleaning and preprocessing the data

Text preprocessing represents a critical foundational stage in data analysis, and in this step we perform fundamental actions for text cleaning. Firstly, we convert all the tweets to lower case, eliminating case variations and ensuring consistent word treatment. This prevents algorithms from treating "Hello" and "hello" as different words. Then, we remove user mentions (@username) and retweet markers (RT), web URLs, punctuation and numeric characters, because they do not provide any significant meaning to the text analysis process. Afterwards, we remove common words like "the", "is", "at" that don't carry significant meaning. Next steps are tokenization and lemmatization. Tokenization is the process of breaking down large blocks of text such as paragraphs and sentences into smaller, more manageable units. Lemmatization reduces words to their base dictionary form, understanding contextual nuances. Finally, we delete all the duplicates from the dataframe, as well as delete the same words in the tweets after lemmatization. Overall, the preprocessing includes the following stages: **lowercase conversion**, **noise removal**, **stopword elimination**, **tokenization, lemmatization**, **and duplicate deletion.** By methodically cleaning and normalizing text, we reduce data complexity, minimise irrelevant information and improve algorithm performance. [1][2]

2.2 Feature Selection and Extraction

Through a range of models discussed in section 3, we mainly focus on the different kinds of information carried in the textual feature of the tweets. Inspired by [3], we consider both word level and bin level representations. On the one hand, sequences of words at the tweet level may carry important information on intricate linguistic patterns. These are processed using Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. On the other hand, given the large size and noisy nature of the dataset, aggregating features in each bin can mitigate noise and capture broader topics. We classify average word embeddings for each bin using a Multi-Level Perceptron (MLP) and classical machine learning models.

We also explore additional features such as tweet volume per bin, which correlates with event occurrence (up to 0.53 for match number 17). However this feature shows high variation across games and does not improve model performance, whether in classical ML models with bin-level representation, or in sequential models. This is true for both instantaneous volume and change in volume from the previous bin.

Similarly, we test the timestamp feature to prioritize recent tweets in each bin, aiming to eliminate noise from earlier irrelevant tweets before event occurrence. However we find that limiting tweets based on their timestamp leads to reduced performance across models due to data loss. As a result, both features are excluded from our final models.

3 Model Choice, Tuning and Comparison

3.1 Classifying individual tweets with LSTMs and CNNs

We begin by applying LSTMs and CNNs to classify individual tweets. LSTMs are effective at capturing sequential dependencies in text, while CNNs perform well in tasks like sentiment analysis and topic categorization [4], both of which are relevant to our task.

For preprocessing, we build a vocabulary using the 10,000 most frequent words in the dataset. Tweets are converted into integer sequences based on this vocabulary. Out-of-vocabulary words are ignored. The most frequent words in the vocabulary are assigned a lower integer value to reflect the smaller amount of information they carry. Sequences are padded to a uniform length of 60 words, the maximum length after preprocessing.

For both models, we use randomly initialised embeddings directly integrated as a first layer and learned in the models. As running individual tweets in LSTMs and CNNs is costly, we work on small random subsets of the data to tune the hyperparameters of our two models, starting with 5% of each bin and increasing gradually. During this process, three games are used as testing, and the rest for training. We sample hyperparameter combinations for vocabulary size, number of layers, number of LSTM units or kernels, dropout rate, batch size and learning rate. The number of batches is chosen using early stopping to prevent overfitting. We try each hyperparameter combination on multiple subsets of the data with varying sizes. The final selection is made based on stability, generalisation ability and training time.

Ultimately, running individual tweets through LSTMs and CNNs is costly in time so we choose two simple architectures: 1 embedding layer, 1 LSTM or convolutional and max pooling layer, a dense layer, a dropout layer and another dense layer. Tweet predictions are assigned with a threshold of 0.5 and are aggregated to bin predictions by majority vote. Despite LSTMs being more complex, both models perform similarly, reaching an accuracy of around 0.68 for the labelled test data. However, when generalized to the Kaggle test data, both models drop to around 0.63 accuracy. Therefore, we explore a new aggregated approach to mitigate the noise in the data.

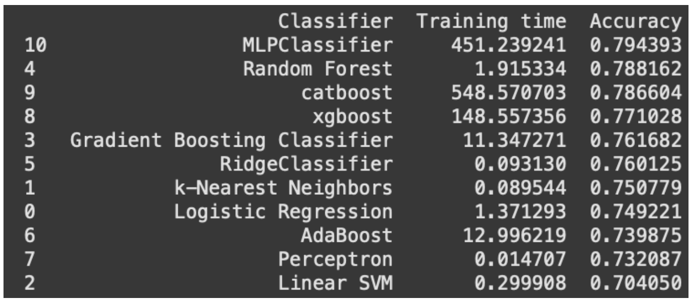
3.2 Bin-level aggregation with MLPs and classical machine learning

Given the limitations of tweet-level classification, we shift focus to bin-level aggregation, treating the entire bin as a unit for classification. We create a single embedding for each bin that is the average embedding over all the words in the bin, using GloVe's Twitter-200 embedding model. This approach should be more efficient and more robust to the noise of our dataset.

We investigate a diverse range of algorithms, including traditional methods like Logistic Regression and k-Nearest Neighbors, ensemble techniques like Random Forest and Gradient Boosting, and more advanced approaches like XGBoost, CatBoost, and a Multi-Layer Perceptron (MLP), a type of Neural Network.

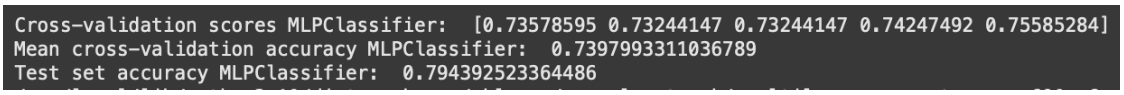
To optimize our models' performances, we employ a systematic grid search technique for the best models. We search for the optimal activation functions, learning rates, number of hidden layers where applicable, and solvers. To mitigate the risk of overfitting, a common issue where a model becomes too specialized to the training data, we implement cross-validation. This technique involves dividing the dataset into multiple folds, training the model on a subset of the data, and evaluating its performance on the remaining fold. By averaging the performance across multiple folds, we obtain a more reliable estimate of the model's generalization ability.

The results reveal that the MLP classifier achieves the highest accuracy at 79.44% only with specification of max_iter=500, after which we get the following best hyperparameters from grid search: "relu" activation, 50 hidden layers, adam solver, constant learning rate with alpha of 0.01, however on testing data the first option performed better. MLP outperforms all other models (see Figure 1). Its stable cross-validation results, pictured in Figure 2, demonstrate its generalisation ability and its strong capability of learning complex relationships within the data. However, its training time is significantly higher compared to other models in this section, highlighting a trade-off between accuracy and computational efficiency.



```
                      Classifier  Training time  Accuracy
10                 MLPClassifier     451.239241  0.794393
4                  Random Forest       1.915334  0.788162
9                       catboost     548.570703  0.786604
8                        xgboost     148.557356  0.771028
3   Gradient Boosting Classifier      11.347271  0.761682
5                RidgeClassifier       0.093130  0.760125
1            k-Nearest Neighbors       0.089544  0.750779
0            Logistic Regression       1.371293  0.749221
6                       AdaBoost      12.996219  0.739875
7                     Perceptron       0.014707  0.732087
2                     Linear SVM       0.299908  0.704050
```

Figure 1. Accuracy of the classification models and training time in seconds



```
Cross-validation scores MLPClassifier:  [0.73578595 0.73244147 0.73244147 0.74247492 0.75585284]
Mean cross-validation accuracy MLPClassifier:  0.7397993311036789
Test set accuracy MLPClassifier:  0.794392523364486
```

Figure 2. MLP Cross-Validation results with 5 Folds

4 Results

Finally, we choose the **MLP Classifier** as a final model and achieve an accuracy of 0.73828 on Kaggle data with our optimized parameters. We conclude that focusing on broader bin-level patterns, while leveraging deep architecture to capture intricate relationships within the data, is the most effective in our tweet classification task.

Reference list

[1] Desilva, M. (2021, August 26). Preprocessing Steps for Natural Language Processing (NLP): A Beginner's Guide. Medium. Retrieved from https://medium.com/@maleeshadesilva21/preprocessing-steps-for-natural-language-processing-nlp-a-beginners-guide-d6d9bf7689c9

[2] Bibsgaard, S., & Kjaerulff, U. (2010). Comparison of text preprocessing methods for document classification. Natural Language Engineering, 16(4), 509-553.

[3] Bekoulis, G., Deleu, J., Demeester, T., & Develder, C. (2019). Sub-Event Detection from Twitter Streams as a Sequence Labeling Problem. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP 2019), 137-146. ACL Anthology. Retrieved from https://aclanthology.org/N19-1081.pdf.

[4] Zhang, Y. & Wallace, B. C. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. Retrieved from https://arxiv.org/pdf/1510.03820.