Empirical Approach to Generating Live Text for Football Matches

ERRICOS MICHAELIDES - CATALIN IONESCU

University of Bristol

"May the odds be ever in your favour" - Suzanne Collins

I. ABSTRACT

In this project we attempt to test the feasibility of crowd-sourced journalism using machine learning techniques. Specifically, we aim to create a stream of live events about football matches. An important application of this is keeping score, an extremely popular subject on the internet. Existing score-keeping sites require a large number of people following matches on the television or radio to update the scores in real time. We take into advantage the abundance of live information available on the internet about football matches to eliminate this need. We acquire data generated on Twitter from the official account of the English first football league (English Premier League), through the Twitter API. The data is processed and a Naive Bayes Classifier is trained on 407 tweets, labeled by hand with one of the classes [Past, Live]. After some extra features are added to the classifier to improve its performance the classifier is able to distinguish between past and live events with very high accuracies (91.98% as tested with 10-fold Cross-Validation).

II. INTRODUCTION

Living in the Information Age makes analyzing data a very powerful tool. From Amazon to Tesco, every company uses data analytics in order to increase their efficiency and boost their customer's engagement. As Mike Olson, co-founder of Cloudera, suggested during one of his talks, this might not be just a trend, but will probably become wide spread technique. We are most likely going to live a second "industrial revolution". While the first one replaced manual labor with machines, the second one might replace people taking decisions with computers performing data analytics.

We are trying to stay ahead of this trend so we decided to use this research project as an opportunity to test the feasibility of crowd-sourced journalism. We are especially targeting live text football websites and are trying to determine whether we can create a live stream of events based on tweets. Our plan is to index tweets and create a machine learning model which will classify them into one of the two categories we are interested: live or past. The ultimate goal of our approach is to create stories (with a publishable quality) based solely on tweets.

At present, most news articles are created by humans after thorough documentation. Hence, we believe that they miss the most important aspect, which is informing people about events in real-time rather than reporting stories.

Twitter is a microblogging platform which allows users to post text messages known as tweets to update their followers with their findings, thinking and comments on some topics [1]. Tweets have a maximum length of 140 characters and can include special words: references (tagging another Twitter user), hashtags (which acts as a tag) and links (to either a website or an uploaded picture / video). Since it was launched in March 2006, Twitter became one of the most important social media websites with about 500 million tweets every day[2]. When an event occurs, it is generally believed that people who witness this will use social media to share their story.

As Paul Lewis highlights during his TED talk [3], journalists are starting to use crowd-sourcing in order to validate the stories they are publishing.

The London protests event he mentions is a valid argument in favor of the idea that social media may contain vital information about real-time events.

Another strong argument in favour of Twitter having vital data which could help journalists create better and more accurate stories was published by Sakaki et al. 2011[4]. They analysed tweets about the 2011 earthquake which occurred in Japan and attempted to uncover what happens on social media during an emergency situation. This is a hands-on approach which shows that social media contain the signals necessary to identify information about events moments after they occur.

Therefore, we decided that the best way to validate whether data from Twitter can be used as a journalistic source was to start with creating live text for ongoing football matches using tweets as input data.

III. HIGHER LEVEL DESIGN

Our approach was to create 3 entities which will deal with data. First one was a Python set of scripts which will fetch, curate and build the machine learning model. This was part of the pre-processing. During run-time, we will have a script which continually sends requests the Twitter API and after classification, it will push them to a web server. The web server was built using Ruby on Rails. The last entity we built is a mobile app which was developed using the Ionic Hybrid Framework. We have decided that during this iteration in our development, we will follow only one Twitter account (namely @premierleague) as it provides a good amount of tweets and an acceptable ratio between our target classes. Furthermore, it is the official account of the English Premier League. This ensures their tweets are in English and have a certain degree of validity: we are confident that we will acquire meaningful and valuable data.

We decided that a Binary Classifier would be best for our initial approach. Hence, we came up with a split for tweets into two classes: Live and Past.

Live tweets are those tweets which are sent while a game is being played, are about the game in development and have meaningful information embedded in their text.

The Past tweets are those who refer to either historic information (i.e. In 1980, X scored 5 goals against Y), or have information which is completely unrelated to any of the live games at that point. Some examples might include statistics about the league or comments made by various people about games.

IV. DESIGN DETAILS

1) Data Acquisition: Twitter provides a RESTful API for accessing tweets. It grants almost unrestricted access (by allowing 450 requests every 15 minutes). The first script we built was used to check the timeline of the account we targeted. After fetching the raw tweets, we went through the first step of data curation. We only kept the sections relevant to our search from the raw data (i.e. text of the tweet, time of creation, tweet id and the avatar if it is present).

The part that was most interesting for us was the text. The text of the tweet was tokenised and certain tokens were extracted. These included references, hashtags and links. We also extracted sequences which we found useful (for example, '[digit]+ mins' is used to let the audience know the number of minutes elapsed since the start of the game). The rest of the text had all punctuation and numeric characters removed.

2) Simple Naive Bayes: The acquired data is passed to the Model training class to create our chosen classifier: Naive Bayes. This is a probabilistic classifier based on the Bayes rule. Suppose that two random variables A and B are given with probability density $P(B \mid A)$ and $P(B)$ respectively. Then the posterior distribution is obtained by:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)},$$

where $\mathbf{B}$ is the data and $\mathbf{A}$ are the unknown parameters. In other words Bayesian inference derives the posterior probability as a consequence of two antecedents, a prior probability and a likelihood function derived from a probability model for the data to be observed[5]. This likelihood function describes our belief that $\mathbf{B}$ would be the outcome if we knew $\mathbf{A}$ to be true. Inference about the parameters is then obtained from the posterior distribution.

- The *prior probability* is the distribution of the parameters before any data is observed, $P(\mathbf{A})$.
- The *likelihood function* is defined as: $f(b_1, \ldots, b_n \mid \mathbf{A}) = \prod_{i=1}^{n} f(b_i \mid \mathbf{A})$
- The *posterior probability*, $P(\mathbf{A} \mid \mathbf{B})$, is the distribution of the parameters after taking into account the observed data. This is determined by Bayes' rule, which forms the heart of Bayesian inference:

$$posterior \propto likelihood \times prior.$$

Naive Bayesian classification is called naive because it assumes class conditional independence[6]. That is, the effect of an attribute value on a given class is independent of the values of the other attributes. In essence this means, that the appearance of a word in a tweet is independent of the appearance of any other word. Although it is very unlikely that words in a tweet are not related to each other, the assumption simplifies the model as it reduces computational costs. For our first iteration of the classifier we only used a simple bag of words stripping off extra information mentioned before: hastags, links, references and timestamps. The class assigned to a tweet is the one with maximum value for the following:

$$c_{MAP} = \text{a}rgmax_c P(c) \prod_i P(w_i \mid c), \qquad (1)$$

where $P(\mathbf{c})$ is the class prior probability. It is multiplied by the product of all probabilities $P(w_i \mid c)$ of words in the tweet found in the given class' training set.

Our class prior is defined as:

$$P(c_j) = \frac{tweetcount(C = c_j)}{N_{tweets}}.$$

In other words it is equal to the count of tweets in that class over the total number of tweets in the training set.

The likelihood is defined as the product mentioned earlier. Each member of the product looks like:

$$P(w_i \mid c_j) = \frac{count(w_i, c_j)}{\sum\limits_{w \in V} count(w, c_j)}.$$

In order to accommodate for words in the tweet not seen in the training set, we apply the Laplace correction to each member of the product. Had this not been done such a case would yield to likelihood 0. V is the vocabulary of all words seen in the training set.

$$P(w_i \mid c) = \frac{count(w_i, c_j) + 1}{\left(\sum\limits_{w \in V} count(w, c_j) \right) + |V|}.$$

These equations were applied to a training set containing 407 tweets, hand labelled by the authors. 150 of these were identified as Live (classID = 0) and 257 as past (classID = 1). Two files were then created containing the words seen in each class, along with the count of times they were seen and their probability yielded by the equations above. In this way when a new tweet is to be classified the process is faster. The only operation made is to check for its words in each class probability file and copy the probability values there. If a word in the tweet does not occur in the file, we use the Laplace correction as stated above. As the product of these probabilities would be infinitesimal instead of (1) we use:

$$c_{MAP} = \text{a}rgmax_c P(c) \sum_i P(log(w_i \mid c)).$$

The tweet is assigned to the class that maximises the above equation. According to 10-fold cross validation which is described in detail in a later section we obtain accuracy of:

(89.6%, 10.4%), correctly vs incorrectly classified tweets.

In an attempt to explain our quite successful classification, we tried to visualise our data. For each word in both classes we defined the likelihood ratio between the two probabilities. The ratios were grouped according to magnitude, with the ones from the middle and to the left favouring the past class, and from the other side the live class. Bucket 1 contains words having 10:1 ratio favouring past, bucket 2 9:1 and so on. Equivalently bucket 19 favours live, 1:9 compared to past and bucket 20 1:10. Buckets at the two extremes show existence of good class-separating features, whereas middle buckets show small likelihood ratios that are not very useful.

Our results clearly show that there is a number of words that can be thought of as good distinctive features. More than 15 words show strong bias towards each class.

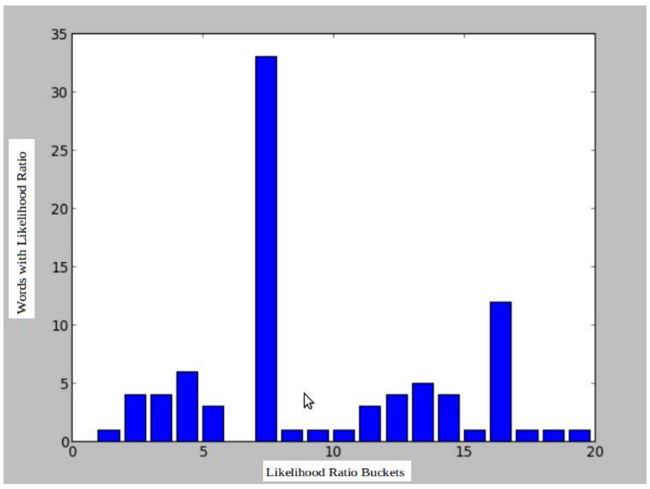


Figure 0: Likelihood Ratios

3) Improvement 1: Dictionary: Although our results were acceptable, we set out to improve the performance of the classifier. Much like a spam filter we could focus on certain special characteristics, increasing their importance in our method[8].

By examining the training set we realised that the account managers publishing the tweets used some key-words for a number of their tweets. For example shortly after a goal is scored in a match, they would tweet this occurrence along with a capitalised string reading ”GOAL”. Similarly when tweeting at times when there are no games going on, they often used keywords like ”REPORT” or ”PREVIEW”. We created a collection of these special words and stored them in a file we refer to as the dictionary. Then for each word in this set we calculated and stored its probability like in simple naive bayes. Now the vocabulary size was much smaller as we only considered the words in the set, so the probabilities were much larger.

When classifying a new tweet we check each word if it exists in the dictionary, we use the log of its probability and sum up over all words. This sum was used in combination with the sum from the previous section. By assigning a weight of 0.6 to the first sum and 0.4 to this new, dictionary-based one and considering the total of this as overall probability, we found an improvement in our results:

(91.69%, 8.31%)

4) Improvement 2: Hashtags: Moving on with our improvements we tried to make use of the very popular feature of tweets: hashtags. Our training set included many hashtags, that without any pre-processing would not give any distinction between our classes. The vast majority of tweets contained at least one hashtag. What was distinctive was the fact that live tweets usually included a 6-character hashtag of the team nicknames joined together. While going through the training set, the number of tweets containing a 6-character hashtag was recorded for each class. This number was lastly divided by the total count of tweets in each class to give another prior probability. We multiplied our result from the previous improvement by this number and classified accordingly. Our results did not improve:

(91.69%, 8.31%)

In a short investigation, the tweets containing hashtags of interest almost always contain other words of interest found in the dictionary, and vice versa. Therefore adding this feature does not influence the classification and was disregarded.

5) Improvement 3: Timestamp: As mentioned above, before applying our model we saved any strings of the form ’[digit]+ mins’. As such strings only appear in tweets when reporting an event in a live match, they had the potential to be very good features. The number of occurrences were counted for each class and a prior was calculated by dividing the count of tweets in each class. This was used like the hashtags feature on top of the bag of words and dictionary and gave the following improved results:

(91.98%, 8.02%).

6) Improvement 4: Links: Our last attempt was to include links as a feature. Both in theory and in practice this did not provide any benefits. The probability of a live tweet from the training set having a link was very similar to that of a past tweet. Namely these were 0.47 - 0.51. The decrease in performance of our classifier reflects the aforementioned facts.

(84.16%, 15.84%).

V. Cross-Validation

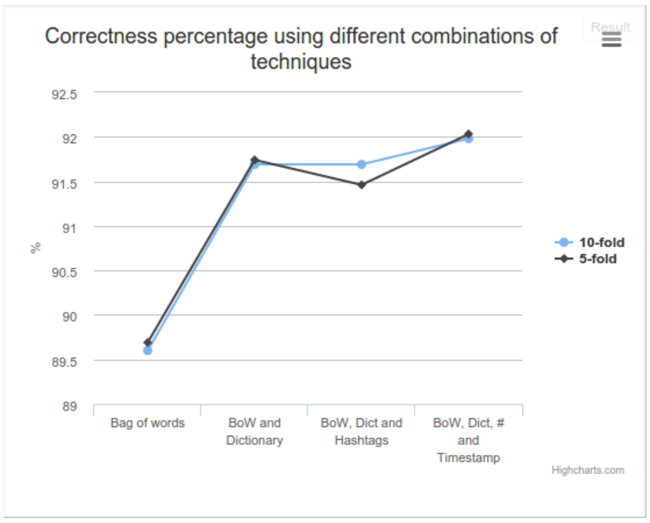
We have used 10-fold Cross-Validation to estimate the correctness of our approach. The implementation was designed to be adaptable mainly due to observing validation techniques used by others (Duan et al. 2010 [8]). They highlighted the usage of 5-fold Cross-validation having a total of 500 tweets labeled for every query they were investigating. This was a logical step because the tweets have only 140 characters (hence the vocabulary will be large while the frequency of words will be low). By splitting the training set into 5 subsets rather than 10, they are ensuring a good ratio between the Train and Test sets. We have decided to use both approaches to test our method. For the Naive Bayes implementation which takes into account just the bag of words (hence, none of the improvements presented in section 4), the results are:

- 10-fold Cross-Validation: Correct: 89.6%; Incorrect: 10.4%
- 5-fold Cross-Validation: Correct: 89.69%; Incorrect: 10.31%

A small improvement in the correction estimation provided by the 5-fold approach, generated by the reasons detailed above.

VI. Weights

Another key aspect of our implementation is the ability to use multiple Naive Bayes priors and then merge the results into a single decision. In order



to achieve this, we are getting each of the probabilities and then apply weights before summing up the probabilities (through our experiments we have applied log to every probability). Hence, determining the perfect weights to join several classifiers becomes a real challenge. We decided to solve this in a programmatic way. For each set of weights, we have a measure of success (i.e. the 10-fold Cross-Validation). We created a class which will run Cross-Validation for all possible combinations of weights, where each weight is in range [0, 2]. Due to the intensity of each Cross-Validation, we decided to use a 0.1 step when going through the range. After several experiments, we decided that there is no use in either increasing the range or reducing the step as there are no foreseeable improvements in our total correctness rate.

We have concluded that for 2 classifiers (one using the simple Bag of Words approach and one using the first improvement (Dictionary)), the best weights are:

- Bag of Words: 0.8
- Dictionary: 0.7

VII. Implementation

A. Fetch - Train - Classify

As described before, we have implemented three code-bases. The first and most elaborate one is the Python set of scripts. All of them were designed using OOP methodologies and were subject of constant refactoring in order to eliminate replicated code. The result was a fairly reusable code-base. There were two stages in this development. The first one was the Training stage. At this step, we wanted to fetch tweets, parse them and create training data (basically files containing a JSON array where each element was a word and had attached the probability). We created one such file for every class.

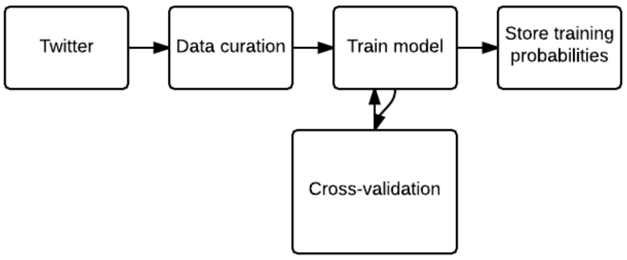


Figure 1: Training pipeline

The second stage was the runtime testing set of scripts. We wanted these to continually fetch data from Twitter, classify each new tweet and assign either a Live or Past class and then push them to the web server if they are classified as Live for storage. If the tweet is classified as Past we will store it locally as we are planning to introduce human validation done posterior to the classification. This will be described in more details throughout the Further Steps section.

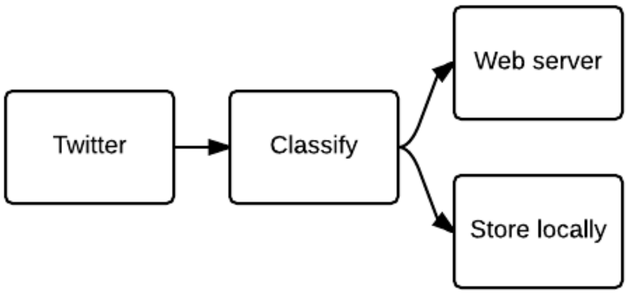
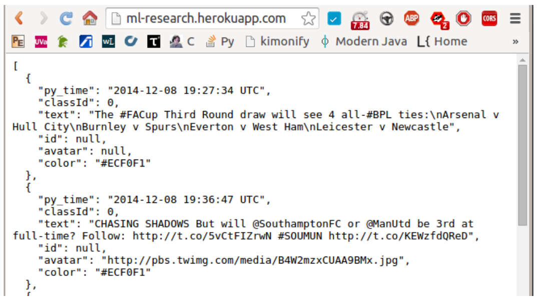


Figure 2: Runtime pipeline

B. Store

Once a tweet is fetched and classified, it will be pushed to the web server. The web server is built using Ruby on Rails and it is designed to be a RESTful API interface for our centralised storage. In an ideal situation, we might have several Runtime instances, running on different machines (say in the AWS cloud) which could fetch tweets from different classes and push them to this centralised database. The web server was created using an MVC structure (Model-View-Controller) such that it will be versatile and robust with further development.

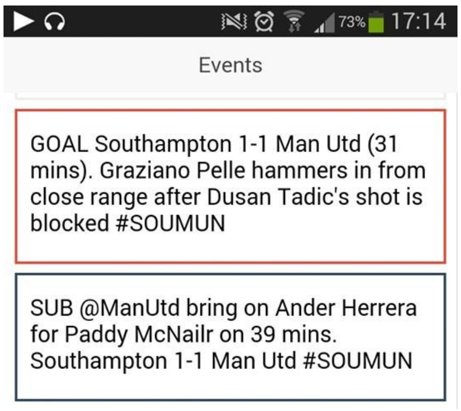


```
[
  {
    "py_time": "2014-12-08 19:27:34 UTC",
    "classId": 0,
    "text": "The #FACup Third Round draw will see 4 all-#BPL ties:\nArsenal v
Hull City\nBurnley v Spurs\nEverton v West Ham\nLeicester v Newcastle",
    "id": null,
    "avatar": null,
    "color": "#ECF0F1"
  },
  {
    "py_time": "2014-12-08 19:36:47 UTC",
    "classId": 0,
    "text": "CHASING SHADOWS But will @SouthamptonFC or @ManUtd be 3rd at
full-time? Follow: http://t.co/5vCtFIZrwN #SOUMUN http://t.co/KEWzfdQReD",
    "id": null,
    "avatar": "http://pbs.twimg.com/media/B4W2mzxCUAA9BMx.jpg",
    "color": "#ECF0F1"
  },
```

Figure 3: GET Tweets Endpoint for the Web Server API

C. Display

The final part of our implementation was the Android Mobile Application built using the Ionic Hybrid Framework. This is a central component for the project as it will be the user interface. It is still at the very beginning of it's development cycle, but we are able to display tweets in a Timeline manner (inspired by the Facebook / Twitter timelines). We are also using color codes for border in order to differentiate events. We are also considering integrating a validation stage in the application (i.e. if you think that a tweet is not about a live game, swipe it to the right).



GOAL Southampton 1-1 Man Utd (31 mins). Graziano Pelle hammers in from close range after Dusan Tadic's shot is blocked #SOUMUN

SUB @ManUtd bring on Ander Herrera for Paddy McNailr on 39 mins. Southampton 1-1 Man Utd #SOUMUN

Figure 4: Mobile interface for Live tweets

VIII. Further Steps

The implementation we described throughout this report was just the first step (the validation) of our idea. The end goal of this project will be to create a versatile, improvable classifier. We will also improve the display of our results by creating a story-like result. Tweets will be grouped according to the live match they describe. We have observed that all tweets referring to a certain match will carry a distinctive hashtag (e.g. Chelsea - Manchester United tweets will have appended the #CHEMUN hashtag). Therefore, we will be able to create real-time updatable stories for every game using the distinguishable hashtag and our Live/Past classifier.

Therefore, our new pipeline will contain some more extra classes. The new system will have 2 tables (stored as JSON arrays - using MongoDB). The first table will contain the tweets which represent the Training set. There will still be one script which will continually create requests to the Twitter API in order to fetch the newest tweets. Once a new tweet is found, it is Classified and added to the Test set. The test set will be visible to the Rails server (to serve as data for the validation step). All tweets in the Test set labelled as Live will be fetched by the mobile app for display.

Another step for improvement will be the correctness of the classifier. Therefore, the Rails web server will also contain a validation step. A human will be able to check the class attributed by our classifier to each tweet. Hence, the human will be able to spot incorrect results (be it false-positives or false-negatives). When the human will correct the class attributed to a tweet, it will be added to the Training set (with the correct label) and all training probabilities will be recomputed. This will ensure that the features which classified this particular tweet badly will not repeat the mistake on future tweets. Therefore, we believe that this approach will create an improvable and versatile classifier.

IX. Conclusion

In this document we have presented an empirical approach to crowd-sourcing live text about football matches. By testing our implementation over several datasets of tweets we have observed its robustness. We also noticed that enlarging the training set yields improved results. We initially trained on 100 tweets and our correctness mean was around 60% while moving to a larger set of 407 we got improvement of almost 32%. Our overall results of 91.98% correct classifications as highlighted by the 10-fold cross validation supports our initial belief that Twitter could be used as a valid journalistic source. The role of people is essential as they could easily spot falsely classified entities and act as the final moderators.

References

[1] http://ebiquity.umbc.edu/paper/html/id/367/Why-We-Twitter-Understanding-Microblogging-Usage

[2] http://www.internetlivestats.com/twitter-statistics/

[3] http://www.ted.com/talks/paul_lewis_crowdsourcing_the_news

[4] http://conferences.sigcomm.org/co-next/2011/workshops/SpecialWorkshop/papers/1569500807.pdf

[5] http://www.cs.columbia.edu/~mcollins/em.pdf

[6] www.researchgate.net/publication/228845263_An_empirical_study_of_the_naive_Bayes_classifier/file/60b7d52dc3ccd8d692.pdf

[7] https://class.coursera.org/nlp/lecture/26

[8] http://www.paulgraham.com/better.html

[9] http://dl.acm.org/citation.cfm?id=1873815