Automatic News Generation Based on Twitter

Ali Alavi[1], Rolf Jagerman[1], and Tsay Kai-En[1]

ETH Zürich, Zürich, Switzerland
`alavis@ethz.ch, {rolfj, tsayk}@student.ethz.ch`

1 Introduction

This report presents the current status of the project **Automatic News Generation Based on Twitter**, for **Big Data** course (code *263-3010-00L*). This project tries to answer the following questions: *Can we automatically generate news headlines based on public twitter posts? Can this method of news generation perform better than the available news agencies, in terms of speed, reliability and so on?*

We tackle this problem by taking the following steps:

1. *Data collection:* Gathering a large set of twitter posts and news headlines
2. *Building a classifier:* Use the news headlines to train a classifier
3. *Labeling the tweets:* Label each tweet using the classifier we previously trained

A big picture of the system is depicted in Figure 1. In the rest of this report we will elaborate on the design and realization of the system.

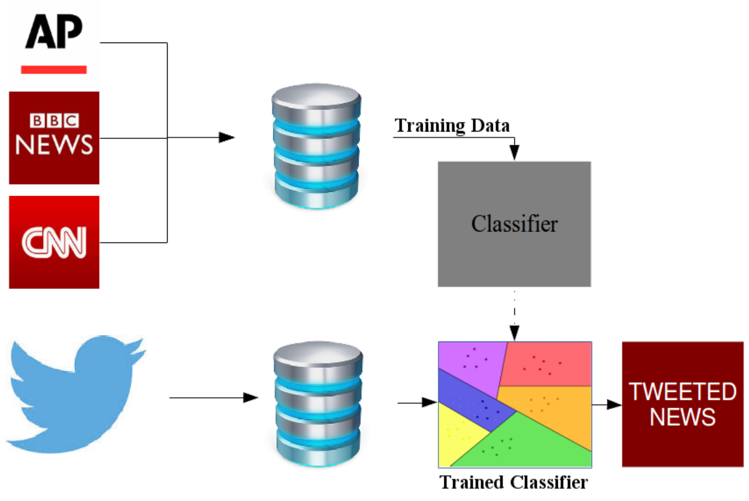


Fig. 1: A big picture of the system

2 Data collection

We used different data sources and data storage systems. All the data collection tasks were performed using Python scripts.

2.1 Data sources

Twitter data Twitter offer different ways of gathering data for different applications [?]. Unfortunately, giving access to the historic data is not one of them. In other words, our access to public data is limited to the most recent tweets. Hence, we decided to collect a large number of the most recent public tweets. Although Twitter Firehose [?] offers access to all public tweets, it requires special permissions to access, which we could not acquire. Luckily, Twitter's *Streaming API* [?] can offer a large number of tweets: it streams a random subset (around one percent) of all the public tweets. This subset is large enough for our purpose. Moreover, the streaming API allows for a much faster collection of tweets in comparison to other APIs such as *REST API*, and also does not have the rate limitations imposed by Twitter on its *REST* APIs [?].

News data We looked into different news agencies (CNN, BBC, Reuters, USA Today, The New York Times,...) and news aggregators (Feedzilla, Google News, Bing Search, ...) in order to collect the required data for training the classifier. Unfortunately, all such sources impose some rather strict limitations on the number of requests and results an agent can send and recieve. As an example, Feedzilla has a limitation of 100 results per request. Bing news is also a commercial product costs of which we cannot afford [?]. Although we managed to collect a good amount of news using The New Your Times [?] and USA Today [?] APIs, we gave them up for a better news source: Twitter accounts of the news agencies.

All news agencies that we looked for have different twitter accounts for different news categories. For the first phase of the project, we chose to use the following news categories:

1. Politics
2. Sports
3. Technology

These categories are chosen due to the following facts:

- There is not much overlap between these news categories (cf. business, technology, finance)
- The frequency of the news generated in these categories are significantly higher that other categories (couple of news items every hour)

We currently use the following Twitter accounts as our news sources: @CNNPolitics, @BBCPolitics, @ReutersPolitics, @BBCSport, @WorldSportCNN, @ReutersSports, @BBCTech, @CNNtech, @ReutersTech. Each account does not have more than some thousands of Tweets, and they do not Tweet more than a couple of news items per hour.

2.2 Data storage

To use the streaming API, the data collecting script needs to send an initial request and then keep listening for the stream of responses. Thus, we need to keep the script running on a server for a long period of time. Moreover, the API publishes more than 2000 tweets per second. This translates to around 20 gigabytes of tweets per day. This imposes a data storage problem, as our resources were rather limited at the first phase of the project. In order to address these issues, we managed to use one of the school's servers to run our script on. This server have around 200 GB of free space, which allows for almost 10 days of continous data collection. Moreover, we managed to use RAR [?] compression to reduce the data size by a factor of 7. Nevertheless, since school servers are not the most reliable type of servers, we decided to take some precautions by setting up a home-based Raspberry Pi [?] server and connecting it to a 1-terabyte external hard disk drive. Using this setup, we are capable of collecting and storing around 350 days of twitter data using the *Streaming API*.

$$\textit{Days of tweet collection} = \frac{\textit{Storage size} \times \textit{Compression ratio}}{\textit{Size of tweets per day}}$$

$$\frac{\textit{1000 GB} \times 7}{\textit{20 GB per day}} = \textit{350 days}$$

3 Design of the system

To select relevant tweets from the massive amount of twitter data, we will have to classify the tweets. By accurately assigning labels to tweets that describe their topics, we can find tweets that belong to our selected categories, i.e. "sports", "technology" or "politics". We used a Python machine learning package, Scikit [?], which significantly simplified development of the classifier. A detailed diagram of the system is seen in Figure 2.

3.1 Classification

Feature extraction Traditionally, one would have to build a dictionary of the entire corpus that is being classified before being able to turn samples into feature vectors that can be used by a classifier. This would require at least one pass over the entire data set. We wish to prevent iterating over the entire data set multiple times, as this would be time consuming. To this end, feature hashing is used to turn texts into feature vectors without requiring a dictionary. The text is tokenized, stemmed and then hashed by a fast non-cryptographic hashing algorithm into a sparse feature vector. By choosing a sufficiently large feature space we minimize the risk of collisions. The generated feature vectors can then be used by a classifier.

Choice of classifier The labeled training data (the news headlines) are processed by a stochastic gradient descent classifier. This type of classifier can converge to a solution without having to load the entire data set into memory. This made it a great choice for this project, as we expect the data to grow beyond the memory bounds of a standard computer.

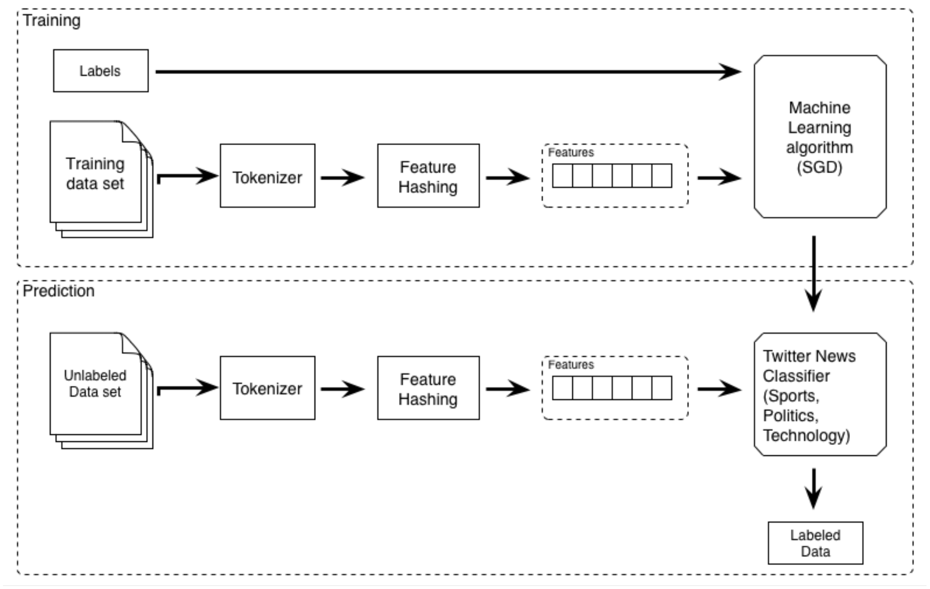


Fig. 2: System diagram

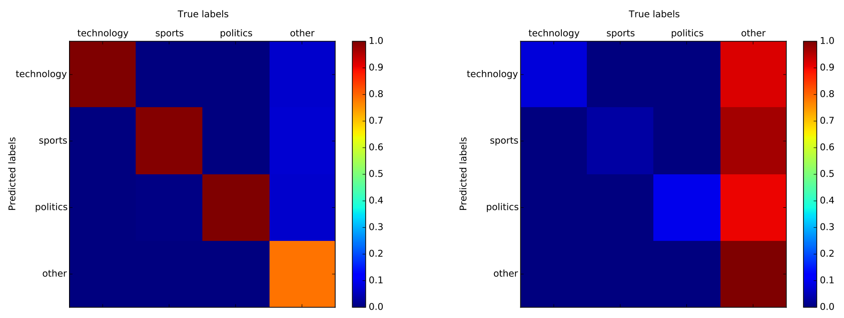
4 Results

To measure the quality of the classifier, we set aside a part of our training data and use that as test data. On this test data we measure several metrics, such as precision, recall and F1-score. These are presented in table 1 and they all display the same behavior: the classifiers are overly conservative in their predictions. This means that the predictions are trustworthy as evidence by a high precision metric. However, a lot of possibly relevant tweets are not given by these classifiers, which results in a low recall. In other words, we have a lot of false negatives.

This behavior becomes even more clear in the confusion matrices displayed in figure 3. The column-normalized confusion matrix shows that when a tweet gets classified into the categories "technology", "sports" and "politics", it is a very good prediction. In contrast, a lot of tweets that should actually be in "technology", "sports" or "politics" are classified as "other", as seen in the row-normalized confusion matrix.

class	precision	recall	f1-score	support
technology	1.00	0.08	0.14	6195
sports	1.00	0.03	0.07	6365
politics	1.00	0.10	0.18	6376
other	0.79	1.00	0.88	65725
avg / total	0.84	0.79	0.71	84661

Table 1: Classification performance over the three trained categories



(a) Column-normalized confusion matrix (b) Row-normalized confusion matrix

Fig. 3: Confusion matrices of the three classifiers

Due to the large amount of data, an overly conservative classifier is not necessarily a bad thing: even though we might be missing a lot of relevant tweets, we are still getting a large amount of useful predictions.

5 Future Works

The data size we worked on for the proof of concept phase is relatively small. For the next step of the project, we want to scale up the data size and add more features to enhance the news classifiers.

We planned to run the massive data set on Spark and use MLlib, which is a Spark implementation of some common machine learning (ML) functionalities for massive data classification. Porting the training and prediction process to Spark will be the challenge that we need to address in this phase. Another challenge will be optimizing the weights of the features, such as numbers of followers and numbers of twitts, to enhance the news classifiers.

References

1. Microsoft. Bing search api. https://datamarket.azure.com/dataset/bing/search, Oct 2014. [Online; accessed 27-October-2014].

2. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
3. The New York Times. Time developer network - apis. http://developer.nytimes.com/docs, Oct 2014. [Online; accessed 27-October-2014].
4. Twitter. Documentation. https://dev.twitter.com/overview/documentation, Oct 2014. [Online; accessed 27-October-2014].
5. Twitter. The streaming api. https://dev.twitter.com/streaming/overview, Oct 2014. [Online; accessed 27-October-2014].
6. Twitter. Twitter firehose. https://dev.twitter.com/streaming/reference/get/statuses/firehose, Oct 2014. [Online; accessed 27-October-2014].
7. Wikipedia. Rar — wikipedia, the free encyclopedia, 2014. [Online; accessed 27-October-2014].
8. Wikipedia. Raspberry pi — wikipedia, the free encyclopedia, 2014. [Online; accessed 27-October-2014].