Auto-Generating Relevant Twitter #Hashtags
A Parallel Optimization to a Natural Language Processing Task

Matthew Hauser
Johns Hopkins University
mhauser5@jhu.edu

Abstract

This paper introduces an approach of using a Naive Bayes classifier to `auto-generate` relevant Twitter hashtags based on a training corpus. This could potentially be useful to help generate more descriptive tags for text. The training data for this model consists of a mix of precollected corpuses of tweets as well as randomly collected tweets using two libraries of Python scripts and the Twitter API. Training and evaluation speeds were parallelized using Spark.

1 Introduction

This paper is based on a machine learning project I worked on in 600.475 last fall. The underlying assumptions about the data are similar, but an entirely new approach is implemented in order to improve the speed of the training and evaluation processes.

Hashtags, or #hashtags, allow users to to"label" statuses and pictures on social media, enabling their content to be searchable by other users. There are no set boundaries on what can be hashtagged–no predefined set of hashtags that all users are confined too. While users typically use hashtags to identify their status or picture with a particular topic (or many topics), hashtags are often used in a way which doesnt truly label their content.

The proposal in this paper is use a Naive Bayes Classifier to "auto-generate" relevant hashtags for Tweets. There is an enormous mass of hand-labeled data: tweets hashtagged by their own users. We use various corpuses retrieved from the Internet, including the Sentiment 140 corpus data[1], the Twitter Political Corpus[2], and randomly-retrieved Tweets using a Python script utilizing the Twitter Stream API[3].

The motivation behind this paper was that due to the conditional independence assumptions of Naive Bayes, there remained agreat deal of parallelism which could be exploited.

2 Naive Bayes

Naive Bayes Classifiers are a family of probabilistic models based on Bayes Theorem with naive assumptions about the data. Bayes Theorem states

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)} \tag{1}$$

Where

1. $P(A)$ represents the prior probability, or the initial belief in A.
2. $P(A|B)$ represents the conditional probability of A given B.
3. $\frac{P(B|A)}{P(B)}$ represents the support B provides for A.

In our case, we are examining the following conditional probability:

$$P(t|\mathbf{W}) = P(t|w_1, w_2, ..., w_N) \tag{2}$$

where t is a hashtag and $\mathbf{W}$ is a sequence of N words $w_1, w_2, ..., w_N$

[1] http://help.sentiment140.com/for-students/
[2] http://www.usna.edu/Users/cs/nchamber/data/twitter/
[3] https://github.com/mdredze/twitter_stream_downloader

We can expand this using Bayes Theorem:

$$P(t|w_1, w_2, ..., w_N) = \frac{P(t)P(w_1, w_2, ..., w_N|t)}{P(w_1, w_2, ..., w_N)} = \frac{P(t, w_1, w_2, ..., w_N)}{P(w_1, w_2, ..., w_N)} \quad (3)$$

The problem is how to calculate and maximize $P(t, w_1, w_2, ..., w_N)$. We *can* approximate $P(w_i|t)$ for any given word w_i with maximum likelihood estimates and counts, namely:

$$P(w_i|t) = \frac{P(w_i, t)}{P(t)} = \frac{count(w_i, t)}{count(t)} \quad (4)$$

Where

1. $count(w_i, t)$ counts the number of times word w_i and tag t are seen together in training data
2. $count(t)$ counts the number of times tag t is seen in training data

We assume that each word, w_i is conditionally independent of all other words. Under these independence assumptions, we are able to calculate $P(t, w_1, w_2, ..., w_N)$. First we expand using the chain rule:

$$\begin{aligned} P(t, w_1, w_2, ..., w_N) &= P(t)P(w_1, w_2, ..., w_N|t) \\ &= P(t)P(w_1|t)P(w_2, ..., w_N|t, w_1) \\ &= P(t)P(w_1|t)P(w_2|t, w_1)P(w_3, ..., w_N|t, w_1, w_2) \\ &= P(t)P(w_1|t)P(w_2|t, w_1)P(w_3|t, w_1, w_2) \\ &\quad *P(w_4, ..., w_N|t, w_1, w_2, w_3) \end{aligned} \quad (5)$$

Under naive assumptions that each word w_i is conditionally independent, we can reduce the following:

$P(w_2|t, w_1) = P(w_2|t)$

$P(w_3|t, w_2, w_3) = P(w_3|t)$

$P(w_i|t, w_{i+1}...w_N) = P(w_i|t)$

$P(w_i|t)$ is a value we can calculate through maximum likelihood estimates. This means $P(t, w_1, w_2, ..., w_N)$ can be calculated as:

$$P(t, w_1, w_2, ..., w_N) = P(t)\prod_{i=1}^{N} P(w_i|t) \quad (6)$$

This means $P(t|\mathbf{W})$ can be calculated as:

$$P(t|\mathbf{W}) = \frac{P(t)}{P(w_1, w_2, ..., w_N)}\prod_{i=1}^{N} P(w_i|t) \quad (7)$$

3 Data

The data used in this trial consists of precompiled corpuses of Tweets as well as automatically downloaded corpuses.

3.1 Precompiled sources of data

- Sentiment140 Corpus: intended for discovering sentiment about a subject through Tweets, consists of 1,048,576 tweet.
- Twitter Political Corpus: intended for classifying whether a given Tweet was political or not, consists of about 4,000 tweets.

3.2 Downloaded Data

Data was generated using the following libraries:

- Python Twitter: a Python wrapper around the Twitter API, used to fetch about 3,000 tweets from 20 randomly-selected users.
- twitter_stream_downloader: a simple Python script used to download a random sample of 570,112 tweets from the Twitter streaming API.

3.3 Processing data

Data was converted into the following format and processed in this way.

- One tweet per line, for reading purposes
- Only tweets with hashtags included in training and evaluating the classifier

4 Spark

The original python implementation of this program consisted entirely of linear programming. Due to the conditional independence and word counting, it seemed like an ideal application for map/reduce programming. Additionally, since there was need for a large amount of shared, read-only data, Spark seemed like the optimal environment for implementation.

4.1 Training

During training, we accumulate two sets of counts which are accomplished through the following mapping

1. $c(word, tag) \rightarrow$ Key: (word, count) Value: 1
2. $c(tag) \rightarrow$ Key: (count) Value: $length$(tweet)

And are reduced using a simple sum as the aggregate operation. It is also necessary to perform a second mapping in order to simply obtain the set of known tags. The counts are transformed into a dictionary object using the Spark API.

4.2 Evaluation

During evaluation, we classify tweets by finding the tag that yields the highest probability of a word sequence upon conditioning on the tag:

$$\arg\max_{tag} P(t = tag) \prod_{N}^{i=1} P(w_i | t = tag) \quad (8)$$

Since this requires a brute force approach over all possible tags, it is highly applicable to map/reduce programming. Each tag is assigned to its own mapper where equation 8 is then calculated. The output of the mapper is the (log) probability of the tweet and the tag occuring together. Each mapper must have access to the probability tables that were constructed, but because during the evaluation phase these tables are read-only, it doesn't create synchronization issues. It maybe worth noting that the probabilites used in evaluation were smoothed in order to prevent novel word/tag pairs from completely prohibiting a given tag.

$$\frac{count(w_i, t)}{count(t)} = \frac{1}{count(t) + V} \quad (9)$$

Where V is equal to the total number of (word, tag) combinations we have seen in training.

4.3 Evaluation

Accuracy is evaluated based on whether each tag in the set of predicted tags is included in the set of original tags that were hashtagged with a given test tweet. If so, the total number of correctly tagged tweets is incremented. Case is ignored, so the tag "#FollowFriday" is considered equivalent to the tag "#followfriday". This is a harsh metric for evaluation because similary tags such as "#ff" and "#followfriday" are deemed distinct tags. Tweets without tags are not considered for evaluation purposes.

5 Code

The code is entire encapsulated in `SparkTwitterTagger.py` and can be run with

```
spark-1.2.0/bin/sparksubmit
SparkTwitterTagger.py train test
local[2]
```

6 Results

The results of the model included an accuracy of about 19 percent which was comparable to that of the linear model. Additionally, the model was able to achieve much higher accuracy (87 percent) when trained on test data. It made fairly impressive improvements over the linear model in terms of speed. Training on a coropus of approximately 500,000 tweets and evaluating on 100,000 took about 2 hours compared to the original 4 hours. This could likely be improved substantially with a more powerful machine which could better exploit the parallelism in the problem (I ran out of AWS credit).

References

[Murty et al.2011] Narasimha Murty, M., Susheela Devi, V. 2011. Pattern Recognition: An Algorithmic Approach.

[Marchetti-Bowick et al.2012] Micol Marchetti-Bowick and Nathanael Chambers. 2012. Learning for Microblogs with Distant Supervision: Political Forecasting with Twitter. *In Proceedings of the European Association for Computational Linguistics*, Avignon, France.

[Go et al.2009] Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter Sentiment Classification using Distant Supervision.

[Jurafsky et al.2008] Jurafsky, Daniel and Martin, James H. 2008. *Speech and Language Processing*,

Acknowledgement

I would like to credit Jade Huang for her original part in the derivation of the model which was the inspiration for this work.