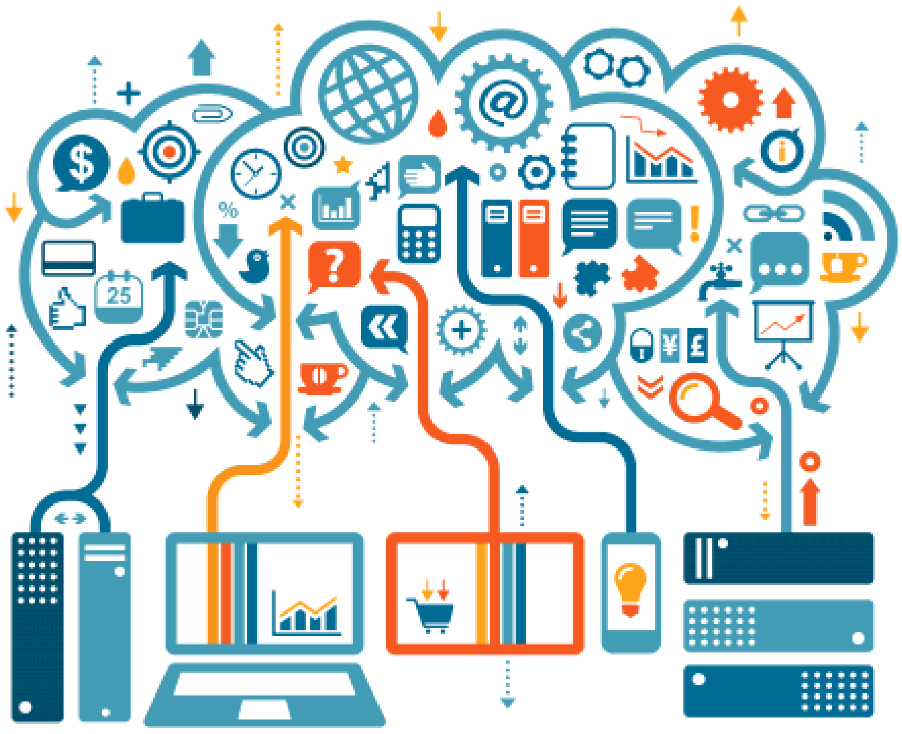
TDT4305: Project Phase 2
Location Estimation of a Tweet

Fredrik Bakken and Tor Arne Hagen

April 24, 2018



NTNU
Norwegian University of
Science and Technology

Brief Description of Delivered Program

All the documentations, source code, and further details about the project can be found on our Github repository: `https://github.com/FredrikBakken/TDT4305_Big-Data-Project`.

Program Startup

One of the requirements for phase two was to setup the program to have three input flags during start up. To solve this, we chose to use an external library known as `argparse`. By using the official documentation for the library, we setup the program start up to have three mandatory flags:

1. `-training` (or `-t`), full path of the training file.
2. `-input` (or `-i`), full path of the input file.
3. `-output` (or `-o`), full path of the output file.

If any of these flags are incorrect or invalid, the program will terminate.

Generate Training Set Sample

In the presentation, a recommendation was to use a sample set of the entire training file. This was solved by the Python Spark function `sample` as follows: `.sample(False, 0.1, 5)`, which is used to return a subset of the entire data set.

Since only two columns in the `geotweets.tsv` file was necessary in the task, we used the `map` function to only take these two columns into account: `.map(lambda pt : (pt[PLACE_NAME], pt[TWEET_TEXT].lower().split(' ')))`, (place is 4 and tweet is 10). The tweet text was set to all lowercase characters and each word was split into a list based (because of the Naive Bayes Classifier's word occurrence count).

Read Input Tweet(s)

The input file is used to represent a random tweet, which the program is going to estimate where is located by comparing with a training set of other actual tweets. Each line in the file is split into a list of words, which then again is appended into a list that holds all the different input tweets (a list of lists, containing different tweet representations).

Total Number of Tweets

One of the variables necessary to calculate the Naive Bayes Classifier is the total number of tweets, which can be found by using the `count` function as: `.count()`.

Probability Calculations

The actual probability calculations consists of two steps, preparations of the data and the actual probability calculations for each location.

Preparation

For the preparation step, `aggregateByKey` is used to perform more complicated calculations. A template is first created to represent the count value for each word in the simulated tweet (zeroValue). For each row of data, the `occurrence_counter` is used as a sequence function to count the number of occurrences of each word for the specific locations, while the number of a specific location is also added up. Then a combination function is initiated to the specific values between partitions.

Next step is to `filter` out all locations which has an occurrence count which is less than 1. This can be done since the Naive Bayes Classifier will result in a probability of 0 for any situation where there is an occurrence count equal to 0.

Calculations

While performing a `map` (to only return necessary data), the probability similarity calculation is executed according to the Naive Bayes formulae:

1. $\frac{|T_c|}{|T|}$, `p = (float(incidents[1]) / float(total_number_of_tweets))`
2. $\frac{|T_{c,w_x}|}{|T_c|}$, `p *= (float(word_count) / float(incidents[1]))`

Find Highest Probable Location

To find the location(s) with the highest probable similarity to the represented tweet (from the input file), a `count` is first executed to make sure that there actually are any tweets with probable similarity (according to Naive Bayes). If there are none, then the value `None` is returned. If there on the other hand are any similar tweets, the highest probability value is found by using the `max` function by: `.max(key=lambda x : x[1])[1]`. The returned probability value can then be used to `filter` out all occurrences of the same probability by: `.filter(lambda x : x[1] == highest_probability).collect()`.

Store Results in File

All locations with the highest probable chance is then stored into the output file. The file is organized by each row representing results from each tweet simulation in the input file. Each row is also split by locations (with tab between) and probability in the end.

Test Results

After the development process was completed, a few simulation tests were ran to check which location had the most probable similarities:

Tweet Text	Location(s)	Probability
Empire	Manhattan, NY	0.00016591085057
No ska oss på fæstival!		
#WhiteHouse	Washington, DC	7.37381558087e-06
Fallingwater	Pennsylvania, USA	3.68690779044e-06
Catedral	Belm, Par	1.84345389522e-05
Iguazu Falls	Foz do Iguau, Paran	4.8511944611e-08
Cabildo	Asuncion, Paraguay	3.68690779044e-06
I love bacon	Avondale, AZ	8.19312842319e-08
I'm drinking coffee. I like it!	Minneapolis, MN	1.44649093188e-14
I want some chocolate	Red Hill, SC	6.32185835123e-09
It's so cold in here!	College Station, TX	2.62297478921e-11

References/Resources

1. https://spark.apache.org/docs/latest/api/python/pyspark.html
2. https://docs.python.org/3/library/argparse.html
3. http://www.learnbymarketing.com/618/pyspark-rdd-basics-examples/