Phase2

April 16, 2018

1 Big Data: Project Phase 2

By Tormod Alf Try Tufteland and Lars Henrik Berg-Jensen

Imports

```
In [1]: from pyspark import SparkContext, SparkConf
        from pyspark.sql import SparkSession, Row
        import os, shutil, datetime
        from collections import Counter
        from operator import add, itemgetter
        from functools import reduce
        import sys
```

Creating Spark Context

```
In [2]: master = "local[4]"
        appName = "phase2"
        conf = SparkConf().setAppName(appName).setMaster(master)
```

```
In [3]: sc = SparkContext(conf=conf)
```

```
In [4]: sc.setLogLevel("WARN")
```

1.0.1 Methods for reading dataset, input tweet and saving result

```
In [5]: def takeTrainingAndReturnList(input_path="data/geotweets.tsv"):
            rdd = sc.textFile(input_path)
            return rdd.map(lambda x: x.split('\t'))
```

```
In [6]: def takeInputAndReturnList(input_path='data/input1.txt'):
            with open(input_path, 'r') as f:
                input_tweet = f.readlines()
                input_tweet_list = [i.rstrip().split(' ') for i in input_tweet]
            return list(map(str.lower, input_tweet_list[0]))
```

```
In [7]: def saveResultToFile(result, output_path="results/output_phase2.tsv"):
            result_rdd = sc.parallelize(result).map(lambda x: '{}\t{}'.format(x[0], x[1]))
            if os.path.isdir(output_path):
                shutil.rmtree(output_path)
            result_rdd.coalesce(1).saveAsTextFile(output_path)
```

1.0.2 Methods for computing number of total tweets and tweets per place

```
In [8]: def getTotalTweetCount(rdd):
            return rdd.count()
```

Counts tweets per place and loads the result as a dictionary in the driver's memory for easy access

```
In [9]: # Computes a tweet count per place map
        def getTweetsPerPlaceCount(place_and_tweets):
            tweets_per_place_count = place_and_tweets.map(lambda x: (x[0], 1)).countByKey().item
            return sc.parallelize(tweets_per_place_count).collectAsMap()
```

1.0.3 Method for counting number of tweets per place containing one of the input tweet words

For each key (place), convert the tuple value to a dictionary containing the input tweet words as keys, and their respective occurences in tweets as values. Counting the tweet words per place, and then combining the result by key.

```
In [10]: #Input: tweets_by_city rdd: [('place1', ['word1', 'word2', 'word3'...]), ...] and input
         #Output: Tuples with place as key and a dictionary containing count for every word from
         def getTweetWithWordByCityCount(tweets_by_city, input_tweet_list):
             # Helper methods used in combineByKey
             def addIfWordInTweet(word, value, tweet_list):
                 if word in tweet_list:
                     value += 1
                 return value

             def to_dict(tweet_list):
                 counter_dict = {k: 0 for k in input_tweet_list}
                 return counter_dict

             def add(counter_dict, tweet_list):
                 return {k: addIfWordInTweet(k, v, tweet_list) for k, v in counter_dict.items()}

             def merge(dict1, dict2):
                 new = {**dict1, **dict1}
                 return new

             counted_tweets_with_word_by_city = tweets_by_city.combineByKey(to_dict, add, merge)
             return counted_tweets_with_word_by_city
```

1.0.4 Method for computing the probability for a single place, based on the Naive Bayes classifier

```
In [11]: # Calculate probability for a single place
         def calculateProbability(place, data_dict, nr_tweets_place, total_tweet_count):
             initial_value = nr_tweets_place / total_tweet_count
             probability = reduce(lambda x, value: x * value / nr_tweets_place, data_dict.values
             return probability
```

1.0.5 Method for combining probability of all places into a new list and then sorting it in descending order

```
In [12]: # Calculate probabilities for all places and sort
         def findProbabilitiesAndSort(counted_tweets_with_word_by_city, tweet_per_place_count_ma
             probabilities = counted_tweets_with_word_by_city\
                     .map(lambda x: (x[0], calculateProbability(x[0], x[1], tweet_per_place_coun
             sorted_probabilities = sorted(probabilities.collect(), key=lambda x: x[1], reverse=
             return sorted_probabilities
```

1.0.6 Wrapper function to make the estimations for each place and then returns the most probable place(s)

```
In [13]: # Wrapper method and outputs the most probable place(s)
         def estimatePlaces(rdd_list, input_tweet_list):
             place_and_tweets = rdd_list.map(lambda x: (x[4], x[10]))\
                                 .mapValues(lambda x: x.split(' '))

             total_tweet_count = getTotalTweetCount(rdd_list)
             tweet_per_place_count_map = getTweetsPerPlaceCount(place_and_tweets)
             tweets_by_city = place_and_tweets.map(lambda x: (x[0], list(map(str.lower, x[1]))))
             counted_tweets_with_word_by_city = getTweetWithWordByCityCount(tweets_by_city, inpu
             sorted_probabilities = findProbabilitiesAndSort(counted_tweets_with_word_by_city, t

             result = []
             top_probability = sorted_probabilities[0][1]
             if top_probability != 0:
                 for place in sorted_probabilities:
                     if place[1] == top_probability:
                         result.append(place)
                     else: break
             return result
```

1.0.7 Fetch training data and input tweet, estimate the places and save the result to file

```
In [14]: rdd_list = takeTrainingAndReturnList()
         input_tweet_list = takeInputAndReturnList()
         places = estimatePlaces(rdd_list, input_tweet_list)
         saveResultToFile(places)
```