Project_Phase1

March 19, 2018

1 Big Data: Project phase 1

2 Data Analysis with Spark

Imports

```
In [2]: from pyspark import SparkContext, SparkConf
        from pyspark.sql import SparkSession, Row
        import os, shutil, datetime
        from collections import Counter
        from operator import add, itemgetter
```

Creating Spark Context

```
In [3]: master = "local[4]"
        appName = "phase1"
        conf = SparkConf().setAppName(appName).setMaster(master)

In [4]: sc = SparkContext(conf=conf)

In [5]: sc.setLogLevel("WARN")
```

Creating RDD from geotweets.tsv

```
In [6]: rdd = sc.textFile("data/geotweets.tsv")
```

Creating a sample RDD for testing

```
In [7]: sampled_rdd = rdd.sample(False, 0.1, 5)
```

Creating RDD arrays by splitting on tabs

```
In [8]: rdd_list = rdd.map(lambda x: x.split('\t'))

In [9]: sampled_rdd_list = sampled_rdd.map(lambda x: x.split('\t'))
```

3 RDD API Tasks

3.1 Task 1

Counting number of tweets by number of elements in RDD

```
In [12]: number_of_tweets = rdd.count()
```

Counting number of users by extracting username column using "map" and counting distinct names

```
In [13]: number_of_users = rdd_list.map(lambda x: x[6]).distinct().count()
```

Same procedure for countries

```
In [14]: number_of_countries = rdd_list.map(lambda x: x[1]).distinct().count()
```

Same procedure for places

```
In [15]: number_of_places = rdd_list.map(lambda x: x[4]).distinct().count()
```

Same procedure for languages

```
In [16]: number_of_languages = rdd_list.map(lambda x: x[5]).distinct().count()
```

Computing min/max latitude and longitude by extracting appropriate columns by mapping and then reducing using min/max methods

```
In [17]: minimum_latitude = rdd_list.map(lambda x: float(x[11])).reduce(lambda a, b: min(a,b))

In [18]: minimum_longitude = rdd_list.map(lambda x: float(x[12])).reduce(lambda a, b: min(a,b))

In [19]: maximum_latitude = rdd_list.map(lambda x: float(x[11])).reduce(lambda a, b: max(a,b))

In [20]: maximum_longitude = rdd_list.map(lambda x: float(x[12])).reduce(lambda a, b: max(a,b))
```

Extracting tweet texts

```
In [21]: tweet_text = rdd_list.map(lambda x: x[10])
```

Computing number of characters in tweets by mapping to length of tweet, and then calculating the mean value of RDD

```
In [22]: tweet_in_characters = tweet_text.map(lambda x: (len(x)))
         average_tweet_in_characters = tweet_in_characters.mean()
```

Same procedure, but first splitting the tweets on space character to get length of words

```
In [23]: tweet_in_words = tweet_text.map(lambda x: len(x.split(' ')))
         average_tweet_in_words = tweet_in_words.mean()
```

Combining results by parallelizing to RDD and writing to file

```
In [24]: results = sc.parallelize([number_of_tweets, number_of_users,\
                                   number_of_countries, number_of_places,\
                                   number_of_languages, minimum_latitude,\
                                   minimum_longitude, maximum_latitude, maximum_longitude,\
                                   average_tweet_in_characters, average_tweet_in_words])
         results = results.coalesce(1)
         resultsPath = 'results/result_1.tsv'
         if os.path.isdir(resultsPath):
             shutil.rmtree(resultsPath)
         results_tsv = results.saveAsTextFile(resultsPath)
```

3.2 Task 2

Creates new RDD by MapReduce, counting number of tweets per country

```
In [25]: tweets_per_country = rdd_list.map(lambda x: (str(x[1]), 1)).countByKey().items()
```

Sorts rdd twice. First alphabetically ascending on country name, then numerically descending on number of tweets. We can do this since the sorts are stable, hence the order between records with same key is preserved

```
In [26]: tweets_per_country_sorted = sorted(tweets_per_country, key=lambda x: x[0])
         tweets_per_country_sorted = sorted(tweets_per_country_sorted, key=lambda x: x[1], reve
```

Saving results as RDD and formatting to correct format

```
In [28]: result_task2_rdd = sc.parallelize(tweets_per_country_sorted)
         result_task2 = result_task2_rdd.map(lambda x: '{}\t{}'.format(x[0],x[1]))
```

Writing results to text file

```
In [29]: resultsPath = 'results/result_2.tsv'
         if os.path.isdir(resultsPath):
             shutil.rmtree(resultsPath)
         result_task2.coalesce(1).saveAsTextFile(resultsPath)
```

3.3 Task 3

Getting countries with 10 tweets or less by filtering results from task 2

```
In [30]: countries_under_10 = result_task2_rdd.filter(lambda x: x[1] < 11)
```

Mapping original rdd to two new RDDs, keyed by country name and latitude and longtitude values, respectively

```
In [31]: countries_with_lat = rdd_list.map(lambda x: (str(x[1]), float(x[11])))
         countries_with_lon = rdd_list.map(lambda x: (str(x[1]), float(x[12])))
```

Getting RDDs with latitude and longitude of countries with over 10 tweets by subtracting by key with countries_under_10

```
In [32]: countries_over_10_with_lat = countries_with_lat.subtractByKey(countries_under_10)
         countries_over_10_with_lon = countries_with_lon.subtractByKey(countries_under_10)
```

Method for calculating center coordinate. Input is a list of latitudes or longitudes. Output is center point

```
In [33]: def calculateCenter(listWithCoord):
             return sum(listWithCoord)/len(listWithCoord)
```

Computing centroid latitude and longtitude for each country by grouping coords by key, converting to list and then perform calculateCenter on each list. Then joining the two RDDs to a new RDD with both latitude and longitude centroid

```
In [34]: country_centroid_lat = countries_over_10_with_lat.groupByKey().\
                                mapValues(list).mapValues(calculateCenter)
         country_centroid_lon = countries_over_10_with_lon.groupByKey().\
                                mapValues(list).mapValues(calculateCenter)
         country_centroid_rdd = country_centroid_lat.join(country_centroid_lon)
```

Formatting results

```
In [35]: result_task3 = country_centroid_rdd.map(lambda x: '{}\t{}\t{}'.format(x[0], x[1][0], x
```

Savings results to file

```
In [36]: resultsPath = 'results/result_3.tsv'
         if os.path.isdir(resultsPath):
             shutil.rmtree(resultsPath)
         result_task3.coalesce(1).saveAsTextFile(resultsPath)
```

Using cartoframes library to visualize cartoDB map in notebook

```
In [40]: import cartoframes
         from cartoframes import Layer, BaseMap, styling
         BASEURL = 'https://larshbj.carto.com'
         APIKEY = '299d2d825191b9879da6fc859d1064930f28d061'
         cc = cartoframes.CartoContext(base_url=BASEURL,
                                       api_key=APIKEY)
         cc.map(layers=Layer('result_task3_carto_4',
                             size=7),
                interactive=False)
```

Out[40]:

3.4 Task 4

Method for calculating local time by converting timestamp to UTC and adding timezone offset. Outputs time rounded to the hour

```
In [41]: def getLocalTimeHour(timestamp, offset):
             s = timestamp / 1000.0 + offset
             return str(datetime.datetime.fromtimestamp(s).hour)
```

Method using Python Counter class to calculate 1-hour interval with most tweets.

```
In [42]: def getMaxTweetTimeInterval(hour_list):
             result = Counter(hour_list).most_common(1)
             return result[0]
```

Extracting country name and calculating local time of tweet using getLocalTimeHour

```
In [43]: rdd_task4 = rdd_list.map(lambda x: (str(x[1]), getLocalTimeHour(float(x[0]), float(x[8
```

Calculating max 1-hour interval per country by grouping tweet hours in lists by key and performing getMaxTweetTimeInterval

```
In [44]: # This can be made more efficient using reduceByKey
         country_time_rdd = rdd_task4.groupByKey().mapValues(lambda x: list(x))\
                              .mapValues(lambda x: getMaxTweetTimeInterval(x))
```

Formatting results

```
In [45]: result_task4 = country_time_rdd.map(lambda x: '{}\t{}\t{}'.format(x[0], x[1][0], x[1]
```

Saving results to file

```
In [46]: resultsPath = 'results/result_4.tsv'
         if os.path.isdir(resultsPath):
             shutil.rmtree(resultsPath)
         result_task4.coalesce(1).saveAsTextFile(resultsPath)
```

3.5 Task 5

Method for finding number of tweets of RDD and sorts in descending and alphabetical order

```
In [47]: def findNumberOfTweetsAndSort(rdd):
             result = rdd.map(lambda x: (str(x[4]), 1)).countByKey().items()
             result = sorted(result, key=lambda x: x[0])
             return sorted(result, key=lambda x: x[1], reverse=True)
```

Filterting RDD on country code 'US' and place type 'city' and computing number of tweets and sorts

```
In [48]: rdd_task5 = rdd_list.filter(lambda x: x[2] == 'US' and x[3] == 'city')
         rdd_task5 = findNumberOfTweetsAndSort(rdd_task5)
```

Formatting results

```
In [49]: result_task5_rdd = sc.parallelize(rdd_task5)
         result_task5 = result_task5_rdd.map(lambda x: '{}\t{}'.format(x[0],x[1]))
```

Savings results to file

```
In [50]: resultsPath = 'results/result_5.tsv'
         if os.path.isdir(resultsPath):
             shutil.rmtree(resultsPath)
         result_task5.coalesce(1).saveAsTextFile(resultsPath)
```

3.6 Task 6

Creating RDD from stop words file and converting to list by splitting on line break

```
In [51]: stopwords_rdd = sc.textFile("data/stop_words.txt")
         stopwords_list = stopwords_rdd.flatMap(lambda x: str(x).split('\n'))
```

Creating RDD with a list of words from US tweets

```
In [52]: rdd_task6_tweets = rdd_list.filter(lambda x: x[2] == 'US')\
                              .map(lambda x: str(x[10]))\
                              .flatMap(lambda x: x.split(' '))
```

Filter this RDD on word length, make all words lower case, subtract the stop words and finally MapReduce to count frequency of all words

```
In [53]: task6_freq_words_list = rdd_task6_tweets.filter(lambda x: len(x) >= 2)\
                              .map(lambda x: x.lower())\
                              .subtract(stopwords_list)\
                              .map(lambda x: (x, 1))\
                              .reduceByKey(add)\
                              .collect()
```

Sorts frequency of words in descending order

```
In [54]: task6_freq_words_list_sorted = sorted(task6_freq_words_list, key=lambda x: x[1], reve
```

Save the 10 most frequent words as RDD and format results

```
In [55]: result_task6 = sc.parallelize(task6_freq_words_list_sorted[0:10])\
                              .map(lambda x: '{}\t{}'.format(x[0], x[1]))
```

Save results to file

```
In [56]: resultsPath = 'results/result_6.tsv'
         if os.path.isdir(resultsPath):
             shutil.rmtree(resultsPath)
         result_task6.coalesce(1).saveAsTextFile(resultsPath)
```

3.7 Task 7

Find the 5 cities with highest number of tweets by filtering results from task 5 and keeping the keys/city names. Create new RDD with city name as key and tweet text as value

```
In [57]: five_cities = result_task5_rdd.zipWithIndex()\
                          .filter(lambda index: index[1] < 5).keys()
         tweet_text = rdd_list.map(lambda x: (x[4], x[10]))
```

Join the two previous RDDs to get a RDD of the five cities with tweet. Split tweet texts into word lists and filter and subtract as in task 6

```
In [58]: sub = stopwords_list.map(lambda x: (0, x))
         tweets_by_city = tweet_text.join(five_cities)\
                                    .map(lambda x: (x[0], x[1][0]))\
                                    .flatMapValues(lambda x: x.split(' '))\
                                    .filter(lambda x: len(x[1]) >= 2)\
                                    .map(lambda x: (x[0], x[1].lower()))\
                                    .subtract(sub)
```

Use CombineByKey to generate dictionaries for each city, containing the number of occourences of each word

```
In [59]: def to_dict(word):
             city = {}
             city[word] = 1
             return city

         def add(city, word):
             if word in city:
                 city[word] += 1
             else:
                 city[word] = 1
             return city

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

         counted_words_by_city = tweets_by_city.combineByKey(to_dict, add, merge)\
                 .collect()
```

Join the 10 most common words with a tab and save to file

```
In [60]: common_words = []
         for city in counted_words_by_city:
             sorted_words = sorted(city[1].items(), key=itemgetter(1), reverse=True)[0:10]
             c = []
             for word_tuple in sorted_words:
                 c.append('\t'.join(map(str,word_tuple)))
             d = '\t'.join(c)
             common_words.append((city[0], d))

In [61]: result_task7 = sc.parallelize(common_words)\
                              .map(lambda x: '{}\t{}'.format(x[0], x[1]))

In [62]: resultsPath = 'results/result_7.tsv'
         if os.path.isdir(resultsPath):
             shutil.rmtree(resultsPath)
         result_task7.coalesce(1).saveAsTextFile(resultsPath)
```

4 Task 8

Creating SparkSession

```
In [63]: spark = SparkSession \
             .builder \
             .master("local") \
             .appName("phase1_dataframe") \
```

```
            .config("spark.some.config.option", "some-value") \
            .getOrCreate()
```

Creating Dataframe using the original RDD of geotweets

```
In [64]: parts = rdd.map(lambda l: l.split('\t'))
         tweets = parts.map(lambda x: Row(\
                                          utc_time=x[0],\
                                          country_name=x[1],\
                                          country_code=x[2],\
                                          place_type=x[3],\
                                          place_name=x[4],\
                                          language=x[5],\
                                          username=x[6],\
                                          user_screen_name=x[7],\
                                          timezone_offset=x[8],\
                                          number_of_friends=x[9],\
                                          tweet_text=x[10],\
                                          latitude=x[11],\
                                          longitude=x[12]\
                                          ))
         df = spark.createDataFrame(tweets)
         df.createOrReplaceTempView("tweets")
```

```
In [65]: sql = """
             select count(*) as number_of_tweets,
                 count(distinct(username)) as distinct_users,
                 count(distinct(country_name)) as distinct_countries,
                 count(distinct(place_name)) as distinct_places,
                 count(distinct(country_name)) as distinct_languages,
                 min(latitude) as minimum_latitude,
                 min(longitude) as minimum_longitude,
                 max(latitude) as maximum_latitude,
                 max(longitude) as maximum_longitude
             from tweets
         """


         df_sql = spark.sql(sql)
         df_sql.show()
```

```
+---------------+-------------+-----------------+--------------+-----------------+--------
|number_of_tweets|distinct_users|distinct_countries|distinct_places|distinct_languages|minimum_
+---------------+-------------+-----------------+--------------+-----------------+--------
|        2715066|        499822|               70|         23121|               70|      -0
+---------------+-------------+-----------------+--------------+-----------------+--------
```