ΠΡΟΧΩΡΗΜΕΝΑ ΘΕΜΑΤΑ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Machine Learning - Ομαδοποίηση δεδομένων με εκτέλεση του k-means αλγόριθμου

Χειμερινό εξάμηνο 2019-20 - Ροή Λ

Αντωνιάδης, Παναγιώτης
el15009@central.ntua.gr

"AI is akin to building a rocket ship. You need a huge engine and a lot of fuel. The rocket engine is the learning algorithms but the fuel is the huge amounts of data we can feed to these algorithms."

– Andrew Ng, *Computer scientist and Statistician*

1 Περιγραφή Δεδομένων

Τα δεδομένα που χρησιμοποιήσαμε είναι πραγματικά και αφορούν σε διαδρομές taxi στην Νέα Υόρκη. Οι δοθείσες διαδρομές των taxi έγιναν από τον Ιανουάριο εώς το Ιούνιο του 2015 και υπάρχουν διαθέσιμες online εδώ. Λόγω των περιορισμένων πόρων που έχουμε στη διάθεσή μας επεξεργαστήκαμε μόνο ένα υποσύνολο μεγέθους 2 GB. Τα δεδομένα αυτά περιέχουν 13 εκατομμύρια διαδρομές, που πραγματοποιήθηκαν το Μάρτιο του 2015 και βρίσκονται εδώ. Στο συμπιεσμένο αρχείο, περιλαμβάνονται δύο comma-delimited αρχεία κειμένου (.csv) που ονομάζονται: yellow_tripdata_1m.csv και yellow_tripvendors_1m.csv. Το πρώτο αρχείο περιλαμβάνει όλη την απαραίτητη πληροφορία για μια διαδρομή και είναι αυτό που μα ενδιαφέρει στη συγκεκριμένη άσκηση. Το αρχείο των TripData έχει την εξής μορφή:

yellow_tripdata_1m.csv

```
369367789289,2015-03-27 18:29:39,2015-03-27
  19:08:28,-73.975051879882813,40.760562896728516,-73.847900390625,40.7326850
  89111328,34.8
369367789290,2015-03-27 18:29:40,2015-03-27
  18:38:35,-73.988876342773438,40.77423095703125,-73.985160827636719,40.76343
  9178466797,11.16
```

Το πρώτο πεδίο αποτελεί το μοναδικό id μιας διαδρομής. Το δεύτερο (τρίτο) πεδίο την ημερομηνία και ώρα έναρξης (λήξης) της διαδρομής. Το τέταρτο και πέμπτο πεδίο το γεωγραφικό μήκος και πλάτος του σημείου επιβίβασης, ενώ το έκτο και έβδομο πεδίο περιλαμβάνουν το γεωγραφικό μήκος και πλάτος του σημείου αποβίβασης. Τέλος, το όγδοο πεδίο δείχνει το συνολικό κόστος της διαδρομής.

2 Περιγραφή Προβλήματος

Ο σκοπός μας είναι, χρησιμοποιώντας τα παραπάνω δεδομένα, να βρούμε τις top 5 περιοχές επιβίβασης πελατών. Για να το επιτύχουμε αυτό θα εφαρμόσουμε των αλγόριθμο ομαδοποίησης k-means στις συντεταγμένες επιβίβασης (4ο-5ο πεδίο) με k=5 περιοχές. Ο αλγόριθμος αυτός είναι επαναληπτικός και θεωρώντας 5 αρχικά κέντρα με προκαθορισμένες συντεταγμένες, γίνονται δύο βήματα σε κάθε επανάληψη:

1. Ανάθεση σημείων σε περιοχή : βρίσκουμε και αντιστοιχούμε κάθε σημείο επιβίβασης στο κοντινότερο από τα 5 κέντρα.

2. Ανανέωση κέντρων περιοχών : υπολογίζουμε τις συντεταγμένες των 5 νέων κέντρων ως τον μέσο όρο των συντεταγμένων των σημείων που ανατέθηκαν σε κάθε περιοχή.

Ο αλγόριθμος συγκλίνει όταν τα κέντρα δεν αλλάζουν πλέον από επανάληψη σε επανάληψη. Ακολουθεί ψευδοκώδικας της υλοποίησης που μας δώθηκε έτοιμος:

```
points = read_data()
k = 5
MAX_ITERATIONS = 3

# Initialize centroids
centroids = get_k_first_points(points, k)
iterations = 1
while not (iterations > MAX_ITERATIONS) {
    iterations += 1

    # For each point in the dataset, chose the closest centroid.
    # Make that centroid's index the point's label.
    points_and_labels = get_labels(points, centroids)

    # Each new centroid is the mean of the points that have the
    # same centroid's label.
    new_centroids = get_new_centroids(points_and_labels)
    centroids = new_centroids
}
print(centroids)
```

3 Μεθοδολογία

Τα βήματα που πρέπει να ακολουθήσουμε ειναι:

1. **Ανέβασμα αρχείων στο HDFS**

   Το μόνο αρχείο που χρειαζόμαστε είναι το yellow_tripdata_1m.csv. Συνεπώς, αφού το κατεβάσουμε από το αντίστοιχο link και το κάνουμε unzip, το ανεβάζουμε στο hdfs με

   ```
   hadoop fs -put ./yellow_tripdata_1m.csv hdfs://master:9000/yellow_tripdata_1m.csv
   ```

2. **Φιλτράρισμα csv αρχείου**

   Αφού διάβασουμε το αρχείο στο Spark, το 1ο βήμα είναι να κρατήσουμε μόνο το 4ο και 5ο πεδίο από κάθε γραμμή, τα οποία είναι αυτά που χρειαζόμαστε. Επίσης, για διευκόλυνση στις πράξεις αργότερα, κρατάμε τις δύο συντεταγμένες (αφού τις μετατρέψουμε σε float) σε ένα numpy array 2 θέσεων. Επίσης, αφαιρούμε όλες τις γραμμές που έχουν γεωγραφικό μήκος ή πλάτος ίσο με μηδέν καθώς θεωρούνται ως 'dirty'.

3. **Αρχικοποίηση κέντρων**

   Πριν την εφαρμογή του αλγορίθμου, θεωρούμε ως κέντρα τις πρώτες 5 γραμμές του αρχείου.

4. **Ανάθεση σημείων σε περιοχή**

Σε αυτό το βήμα, έχουμε σαν είσοδο τις συντεταγμένες επιβίβασης και θέλουμε να υπολογίσουμε για κάθε μία από αυτές το κοντινότερο κέντρο (διαδικασία map). Για λόγους που θα γίνουν κατανοητοί αργότερα συμφέρει η map να επιστρέφει για κάθε γραμμή μία tuple που θα έχει ως 1ο στοιχείο το index του κοντινοτερου κέντρου και ως 2ο στοιχείο μία tuple της μορφής (συντεταγμένες σημείου, 1). Να σημειωθεί ότι αφού θέλουμε να υπολογίσουμε αποστάσεις συντεταγμένων, χρησιμοποιούμε την απόσταση haversine.

5. **Ανανέωση κέντρων περιοχών**

   Στο σημείο αυτό πρέπει να υπολογίσουμε για κάθε κέντρο τον μέσο όρο των συντεταγμένων που ανατέθηκαν σε αυτό στο προηγούμενο βήμα. Χωρίσουμε αυτήν την διαδικασία σε δύο μέρη: πρώτα θα υπολογίσουμε το άθροισμα των συντεταγμένων και το πλήθος των στοιχείων που ανατέθηκαν στο κάθε κέντρο και, στη συνέχεια, θα διαιρέσουμε τα δύο αυτά μεγέθη για να προκύψει ο μέσος όρος. Στο 1ο βήμα, λοιπόν, θέλουμε να κάνουμε μία λειτουργία για κάθε κέντρο. Αυτός είναι ο λόγος που θέλαμε στο προηγούμενο βήμα μία έξοδος της μορφής (κέντρο, (σημείο, 1)). Εφαρμόζουμε μία διαδικασία reduceByKey η οποία σε κάθε κλειδί-κέντρο αθροίζει μεταξύ τους τα σημεία και τους άσσους και καταληγουμε σε 5 γραμμές της μορφής (κέντρο, (άθροισμα σημείων, πλήθος σημείων)). Το μόνο που έμεινε είναι με μία διαδικασία map να διαιρέσουμε για κάθε γραμμή το άθροισμα των συντεταγμένων με το πλήθος των στοιχείων.

6. **Ανανέωση κέντρων**

   Το τελευταίο βήμα σε κάθε επανάληψη είναι να ανανεώσουμε τα κέντρα με τις νέες τιμές. Πρέπει πρώτα να επιστρέψουμε τα νέα κέντρα στον driver με την συνάρτηση collect() και στη συνέχεια επαναληπτικά να ανανεώσουμε τα 5 κέντρα.

7. **Αποθήκευση αποτελεσμάτων στο hdfs**

   Όταν ολοκληρωθούν και οι 3 επαναλήψεις, μοιράζουμε κατανεμημένα τα τελικά κέντρα με το parallelize() και τα αποθηκεύουμε στο hdfs ως απλό αρχείο κειμένου.

Τα βήματα 4, 5 και 6 επαναλαμβάνονται συνεχώς μέχρι να υπάρξει σύγκλιση. Στην υλοποίησή μας θεωρούμε ότι ότι χρειάζονται 3 μόνο επαναλήψεις του αλγορίθμου ώστε να επιτευχθεί σύγκλιση.

4 Ψευδοκώδικας

Algorithm 1: Map για το φιλτράρισμα της εισόδου

```
map(key, value):
// value: information about a route;
   longtitude = 4th collumn of value
   latitude = 5th collumn of value
emit(key, [longtitude, latitude])
```

Algorithm 2: Map για την απαλοιφή dirty γραμμών

```
map(key, value):
// value: Coordinates of a point;
   if value[0] != 0 and value[1] != 0:
      emit(key, value)
```

Algorithm 3: Map για την ανάθεση σημείων σε περιοχή

```
map(key, value):
// value: Coordinates of a point;
   distances = []
   for each center in centers:
      distances.append(haversine(value, center))
   emit(min_index(distance), (value, 1))
```

Algorithm 4: Reduce για τον υπολογισμό του αθροίσματος και του πλήθους

```
reduce (key, values):
// key: Index of a center;  values: an iterator over tuples in the form (point, 1)
   sum = 0
   size = 0
   for each (point, 1) in values:
      sum += point
      size += 1
   emit(sum, size)
```

Algorithm 5: Map για τον τελικό υπολογισμό των νέων κέντρων

```
map(key, value):
// value: tuple that contains the sum and the size of a center
   sum, size = value
   emit(sum/size)
```

Αναφορές

[1] rdd-programming-guide https://spark.apache.org/docs/latest/rdd-programming-guide.html