2017

Twitter Community Detection

ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ ΠΑΓΚΟΣΜΙΟΥ ΙΣΤΟΥ

ΑΝΑΓΝΩΣΤΟΥ ΑΝΤΩΝΙΟΣ 2268
ΛΑΣΚΑΡΙΔΗΣ ΣΤΕΦΑΝΟΣ 2315
{ANAGNOAD,LASKSTEF}@CSD.AUTH.GR

Περιεχόμενα

Εισαγωγή

Η παρούσα εργασία εκπονήθηκε στα πλαίσια του μαθήματος «Πληροφοριακά Συστήματα Παγκοσμίου Ιστού» του 7ου εξαμήνου του τμήματος Πληροφορικής ΑΠΘ.

Συγκεκριμένα, σκοπός της είναι η συλλογή και επεξεργασία tweets για μία θεματολογία σε χρονικό διάστημα 3 ημερών καθώς και η ανάλυση της ομοιότητας μεταξύ χρηστών βάσει συγκεκριμένων πεδίων.

Το τεχνικό μέρος της εργασίας αναπτύχθηκε σε Java 1.8, με τη συλλογή από το Twitter Streaming API [1] να χρησιμοποιεί Spark Streaming (v. 2.0.1) [2], ενώ τα υπόλοιπα τμήματα συμβατική Java με χρήση βιβλιοθηκών. Για την αποθήκευση των πρωτογενών δεδομένων, χρησιμοποιήθηκε MongoDB [3] (v3.4.0). Στη συνέχεια εξήχθησαν οι χρήσιμες πληροφορίες σε σχεσιακή βάση δεδομένων PostgreSQL [4] (v. 9.2.18). Για την ανάλυση των γράφων χρησιμοποιήθηκε το πρόγραμμα Gephi [5] (v. 0.9.1). Τέλος, το παραπάνω software εγκαταστάθηκε σε εικονική μηχανή του ~okeanos IaaS [6] με λειτουργικό σύστημα CentOS (v. 7.2.1511).

Εκτέλεση

Για την εκτέλεση της εργασίας, απαιτούνται οι εξής εντολές:

```
################################################################################################################
# > How to Run
################################################################################################################
# Before running, make sure you have MongoDB, PostgreSQL and Spark installed and configured properly.
# The tables to be created in PostgreSQL are included in the sql/*.sql files.

# Clone repository from Github
git clone https://github.com/authprojects/twitter-community-detection.git
cd twitter-community-detection/

# Enter database credentials in src/main/resources/config.properties
# Enter twitter data in src/main/resources/twitter4j.properties

# Build solution jar
sbt assembly

# Gather tweets from Streaming API
spark-submit target/scala-2.11/twitter-community-detection-assembly-1.0.jar --class TwitterDataCollection

# Gather valuable data for processing from Mongo to Postgres
java -cp target/scala-2.11/twitter-community-detection-assembly-1.0.jar MongoToPostgres

# Compute and enter similarities in Postgres
java -cp target/scala-2.11/twitter-community-detection-assembly-1.0.jar SimilarityComputation

# Compute and print NMI
java -cp target/scala-2.11/twitter-community-detection-assembly-1.0.jar NMIComputation
```

Περιγραφή προβλήματος

Ζητάται η συλλογή δεδομένων από το twitter για tweets χρηστών μέσω του Streaming API το οποίο η πλατφόρμα προσφέρει. Πρωτογενώς, είναι αναγκαία η αποθήκευση των δεδομένων σε MongoDB NoSQL database.

Στη συνέχεια, απαιτείται η εξαγωγή των εξής μετρικών από τα συλλεγμένα δεδομένα:

- User
- Hashtag
- URLs
- Retweet
- Mention

και η αποθήκευσή τους σε κατάλληλη βάση δεδομένων.

Εφόσον τα παραπάνω βήματα ολοκληρωθούν επιτυχώς, επόμενο βήμα αποτελεί ο υπολογισμός των μέτρων ομοιότητας μεταξύ των χρηστών βάσει hashtags, urls, retweets και mentions. Προτεινόμενα μέτρα ομοιότητας αποτελούν Cosine Similarity, Jaccard Similarity, Frequency-based Similarity. Στο τέλος, αναμένεται να υπάρχουν 5 πίνακες ομοιότητας ανά μετρική ομοιότητας, 4 για τα επιμέρους πεδία και 1 συνολικός.

Χρησιμοποιώντας τα μέτρα ομοιότητας που υπολογίστηκαν παραπάνω, απαιτείται η μοντελοποίηση των δεδομένων σε μορφή γράφου με το κατάλληλο layout για οπτικοποίηση και η εξαγωγή των ακόλουθων μετρικών μέσα από το περιβάλλον του Gephi:

- Average Clustering Coefficient
- Graph Diameter
- Graph Density

Τέλος, βάσει των κοινοτήτων που εντοπίζονται, αναμένεται ο υπολογισμός του δείκτη NMI μεταξύ διαφορετικών μετρικών ομοιότητας ανά πεδίο.

Συλλογή και Αποθήκευση Πρωτογενών Δεδομένων

Για την συλλογή δεδομένων, επιλέξαμε να χρησιμοποιήσουμε το Spark framework σε περιβάλλον Java προκειμένου να καταστεί εύκολη η συλλογή δεδομένων από το Twitter Streaming API για 3 συνεχόμενες ημέρες.

Συγκεκριμένα, η κλάση `TwitterDataCollection` είναι υπεύθυνη για τη συλλογή των δεδομένων από το Twitter και την απευθείας εισαγωγή αυτών στη MongoDB.

Το Spark Streaming λειτουργεί σε micro-batches, γεγονός που σημαίνει πως 5 δευτερόλεπτα ζητώνται τα tweets για τα hashtags και γίνεται η αποθήκευση των αντικειμένων που επιστρέφονται σε JSON format απευθείας στη βάση δεδομένων. Δεδομένου ότι η MongoDB χρησιμοποιεί BSON (Binary JSON) για αναπαράσταση των δεδομένων εσωτερικά, η εισαγωγή των εισερχόμενων δεδομένων κατέστη ιδιαίτερα εύκολη με χρήση του Java Mongo Client.

Συγκεκριμένα, συλλέχθηκαν tweets 3 ημερών (05/12/2016 – 07/12/2016) για τα ακόλουθα hashtags, βάσει επικαιρότητας:

- #trump
- #obama
- #brexit
- #italyreferendum
- #austrianelection

Μοντελοποίηση Δεδομένων

Για την εξαγωγή των σχετικών με την περεταίρω ανάλυση δεδομένων, χρησιμοποιήσαμε την κλάση `MongoToPostgres` ώστε να επιλέξουμε τα εξής στοιχεία από τα 1000 πρώτα tweets:

- user_id
- hashtag
- url
- retweet_id
- mention_user_id

Συγκεκριμένα, δημιουργήσαμε 4 πίνακες (user_hashtag, user_url, user_mention, user_retweet), όπου ο καθένας συνδέει τον εκάστοτε χρήστη του tweet με το/τα hashtags, url, mention_id, retweet_id που περιέχονται στο tweet αυτό. Στην περίπτωση που ένας χρήστης μιλάει για πολλαπλά hashtags, αυτά εισάγονται ως ξεχωριστές γραμμές στον αντίστοιχο πίνακα.

Δεδομένου του τετραγωνικού κόστους σύγκρισης κάθε χρήστη με κάθε άλλον για υπολογισμό της ομοιότητας μεταξύ τους, προβήκαμε στη μείωση των tweets που αναλύουμε σε 1000 εκ των 578723.

Για την ανάγνωση των δεδομένων από τη MongoDB χρησιμοποιήθηκε ο Java Mongo Client και για την εγγραφή σε Postgres χρησιμοποιήθηκε ο JDBC driver.

Οι λόγοι για τον οποίο επιλέχθηκε μία σχεσιακή βάση δεδομένων για τη μοντελοποίηση της διαθέσιμης πληροφορίας προς ανάλυση είναι:

- Εύκολη προσαρμογή των δεδομένων μας στο αντικειμενο-σχεσιακό μοντέλο
- Ταχύτητα εύρεσης και επιστροφής αποτελεσμάτων
- Υποστήριξη SQL interface
- CAP support (Consistency, Availability, Partition Tolerance)

Επιπλέον, ο μικρός αριθμός δεδομένων καθιστά την Postgres μία από τις καλύτερες διαθέσιμες επιλογές για την αποθήκευση των δεδομένων μας εφόσον δε συναντώνται κωλύματα scalability.

Υπολογισμός Ομοιότητας Χρηστών

Για την αποθήκευση της ομοιότητας χρηστών, δημιουργήθηκαν οι ακόλουθοι πίνακες στην PostgreSQL:

{cosine,jaccard}_similarity_{hashtag,url,mention,retweet}

Η επιλογή των παραπάνω μετρικών βασίστηκε στο ότι έχουμε δομήσει τα δεδομένα μας με τρόπο που να επιτρέπει την εύκολη διανυσματική αναπαράστασή τους.
Υπολογίζουμε για κάθε πεδίο και τις δύο μετρικές, των οποίων και τα αποτελέσματα συγκρίνουμε με το δείκτη NMI στο τελευταίο τμήμα της εργασίας.

Είναι σημαντικό να διευκρυνήσουμε σε αυτό το σημείο πως τα δεδομένα μας δεν είναι δυαδικά, αλλά ουσιαστικά μετράμε counts της εκάστοτε τιμής πεδίου ανά χρήστη. Την πληροφορία αυτή ενσωματώνουμε και στις παρακάτων μετρικές ομοιότητας.

Cosine Similarity

Ορίζουμε το cosine similarity [7] μεταξύ δύο χρηστών A, B:

$$\frac{A \bullet B}{|A| * |B|}$$

Όπου:

- A: το διάνυσμα χρήστη ως προς το πεδίο. Σε κάθε γραμμή του διανύσματος αποθηκεύεται το πλήθος των εμφανίσεων της συγκεκριμένης τιμής του πεδίου. Για παράδειγμα, έστω πως εμφανίζονται στο σύνολο των χρηστών τα hashtags {x, y, z}. Και έστω ακόμα χρήστης u1, με τα εξής tweets: {“tweet1 #x”, “tweet2 #x”, “tweet3 #x”, “tweet4 #z”}. Τότε το διάνυσμα αυτού θα είναι το [3 0 1]. Στην απόθηκευση των διανυσμάτων, οι μηδενικές τιμές παραλείπονται.
- |A|: η νόρμα του διανύσματος χρήστη ως προς το πεδίο.

Υπολογίζουμε το εσωτερικό γινόμενο ως το άθροισμα των γινομένων των κοινών {hashtags,urls,mentions,tweets} των χρηστών A και B.
Υπολογίζουμε το μέτρο των διανυσμάτων ως τη τετραγωνική ρίζα του αθροίσματων των τετραγώνων των μη μηδενικών {hashtags,urls,mentions,tweets} του εκάστοτε χρήστη.

Jaccard Similarity

Ορίζουμε ως Jaccard Similarity [8] μεταξύ δύο χρηστών A, B:

$$\frac{|A \cap B|}{|A \cup B|}$$

Όπου:

- A: Το σύνολο των {hashtags,urls,mentions,tweets} ενός χρήστη.

Ο υπολογισμός και αποθήκευση των ομοιοτήτων μεταξύ χρηστών γίνεται από την κλάση `SimilarityComputation`.

Δημιουργία και Ανάλυση Γράφου

Δημιουργία Γράφου

Για την δημιουργία του γράφου των συνδεδεμένων χρηστών, χρησιμοποιήθηκαν οι μετρικές της ομοιοτήτας που περιγράφηκαν στο προηγούμενο κεφάλαιο.

Συγκεκριμένα στον γράφο των χρηστών, κάθε κόμβος *i* αναπαριστά έναν μοναδικό χρήστη του Twitter, ενώ μία ακμή από τον κόμβο *i* στον κόμβο *j* με βάρος *w* υποδηλώνει πως οι δύο χρήστες συνδέονται μεταξύ τους με ομοιότητα *w*.

Για την εξαγωγή των δεδομένων από την σχεσιακή βάση και την εισαγωγή τους στο Gephi χρησιμοποιήθηκε η εντολή COPY της SQL που δίνει την δυνατότητα εξαγωγής πινάκων σε μορφή CSV.

Υπολογισμός Στατιστικών Μέτρων

Όπως περιγράφηκε και σε προηγούμενο κεφάλαιο, χρησιμοποιήθηκαν δύο μέτρα ομοιότητας (Cosine, Jaccard) με βάση τα πεδία των tweets (hashtags, URLs, mentions, retweets) καθώς και μία συνδιαστική μετρική ανά μέτρο ομοιότητας. Γίνεται, συνεπώς, κατανοητό, πως συνολικά προέκυψαν δέκα γράφοι χρηστών, ανάλογα με το επιλεγμένο πεδίο και μέτρο ομοιότητας, για καθέναν από τους οποίους, ζητείται ο υπολογισμός των ακόλουθων στατιστικών μετρών:

1. Average Clustering Coefficient
2. Graph Diameter
3. Graph Density

Τα αποτελέσματα των στατιστικών αυτών μετρών δίνονται παρακάτω συγκριτικά ανά πεδίο και ανά μέτρο ομοιότητας.

Ομοιότητα με βάση hashtags

	Ομοιότητα Cosine	*Ομοιότητα Jaccard*
Average Clustering Coefficient	0.948	0.948
Graph Diameter	4	4
Graph Density	0.37	0.37

Ομοιότητα με βάση URLs

	Ομοιότητα Cosine	*Ομοιότητα Jaccard*
Average Clustering Coefficient	0.907	0.969
Graph Diameter	4	3
Graph Density	0.358	0.052

Ομοιότητα με βάση mentions

	Ομοιότητα Cosine	*Ομοιότητα Jaccard*
Average Clustering Coefficient	0.963	0.963
Graph Diameter	6	6
Graph Density	0.019	0.019

Ομοιότητα με βάση retweets

	Ομοιότητα Cosine	*Ομοιότητα Jaccard*
Average Clustering Coefficient	0.86	0.96
Graph Diameter	3	3
Graph Density	0.018	0.018

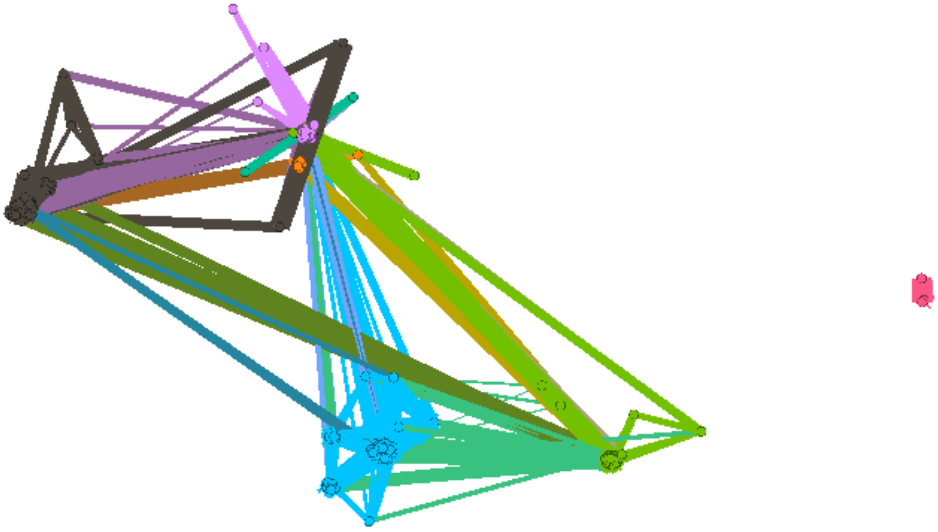
Συνολική ομοιότητα

	Ομοιότητα Cosine	*Ομοιότητα Jaccard*
Average Clustering Coefficient	0.946	0.946
Graph Diameter	5	5
Graph Density	0.295	0.295

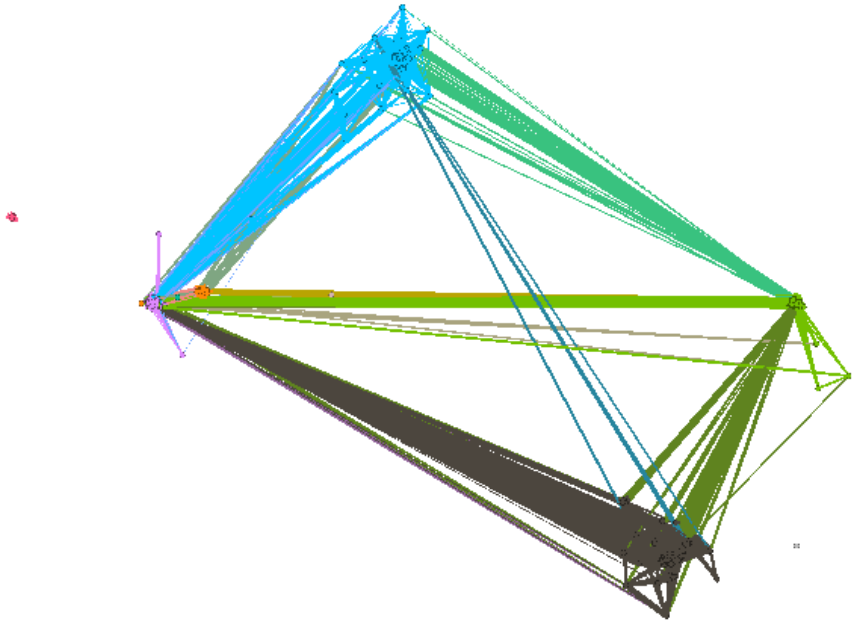
Εύρεση Κοινοτήτων

Για την εύρεση των κοινοτήτων των χρηστών χρησιμοποιήθηκε ο αλγόριθμος του Louvain, ο οποίος είναι υλοποιημένος στο εργαλείο του Gephi. Παρακάτω, παρουσιάζονται χρωματισμένοι οι γράφοι, βάση των κοινοτήτων των χρηστών και διατεταγμένοι με τον αλγόριθμο του OpenOrd.

Εύρεση κοινοτήτων χρηστών χρησιμοποιώντας το πεδίο των hashtags

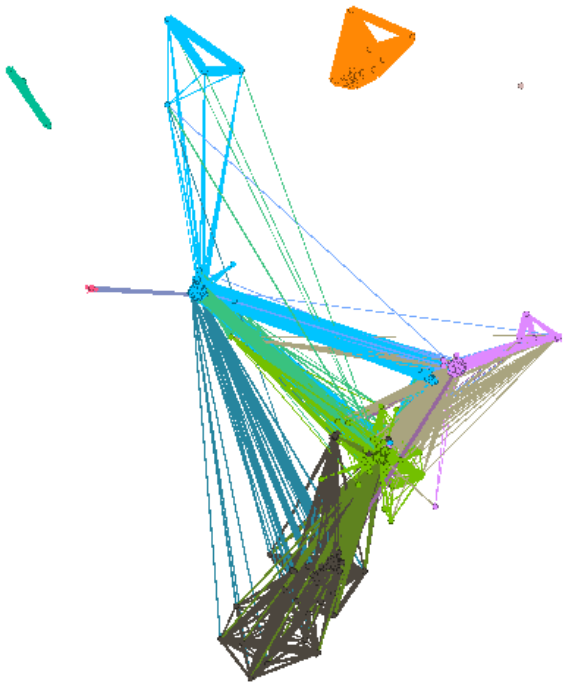


Εικόνα 1Κοινότητες χρηστών με βάση την ομοιότητα συνημιτόνου και του πεδίου των hashtags

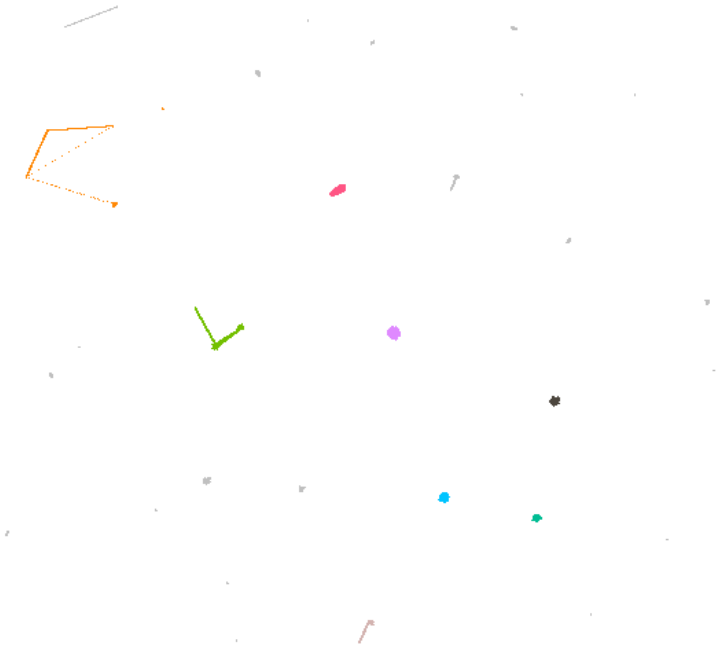


Εικόνα 2Κοινότητες χρηστών με βάση την ομοιότητα Jaccard και του πεδίου των hashtags

Εύρεση κοινοτήτων χρηστών χρησιμοποιώντας το πεδίο των URLs

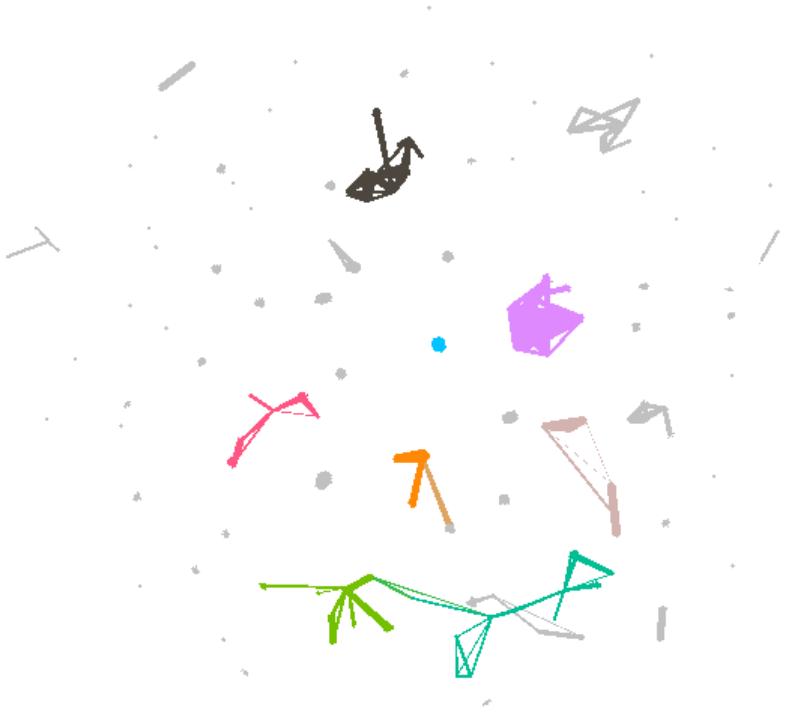


Εικόνα 3Κοινότητες χρηστών με βάση την ομοιότητα συνημιτόνου και του πεδίου των URLs

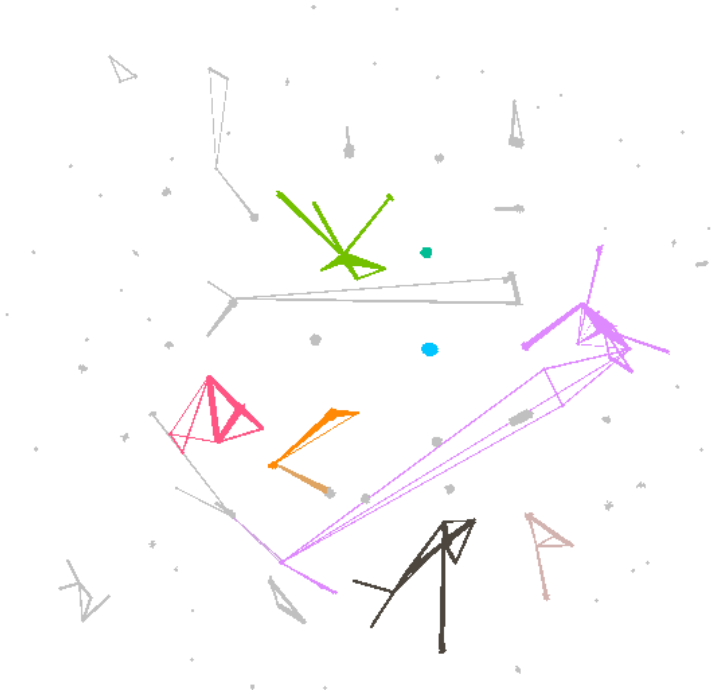


Εικόνα 4Κοινότητες χρηστών με βάση την ομοιότητα Jaccard και του πεδίου των URLs

Εύρεση κοινοτήτων χρηστών χρησιμοποιώντας το πεδίο των mentions



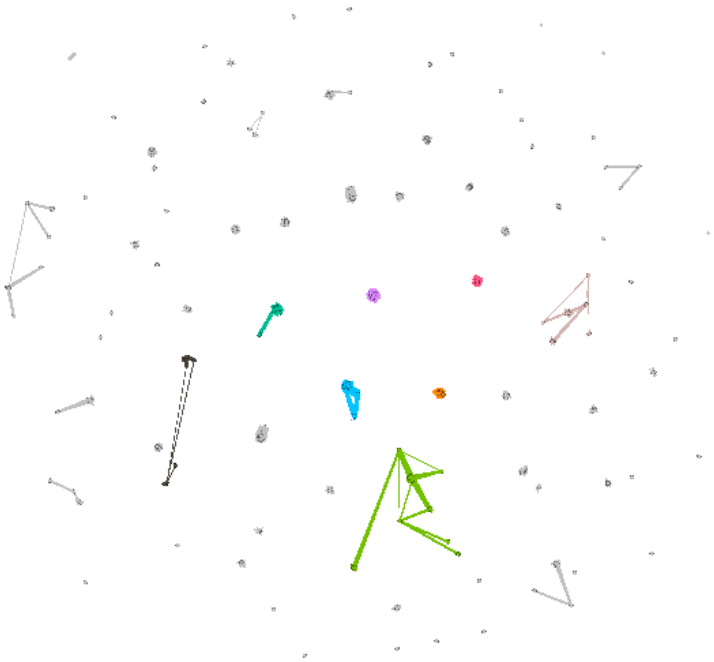
Εικόνα 5Κοινότητες χρηστών με βάση την ομοιότητα συνημιτόνου και του πεδίου των mentions



Εικόνα 6Κοινότητες χρηστών με βάση την ομοιότητα Jaccard και του πεδίου των mentions

Εύρεση κοινοτήτων χρηστών χρησιμοποιώντας το πεδίο των retweets

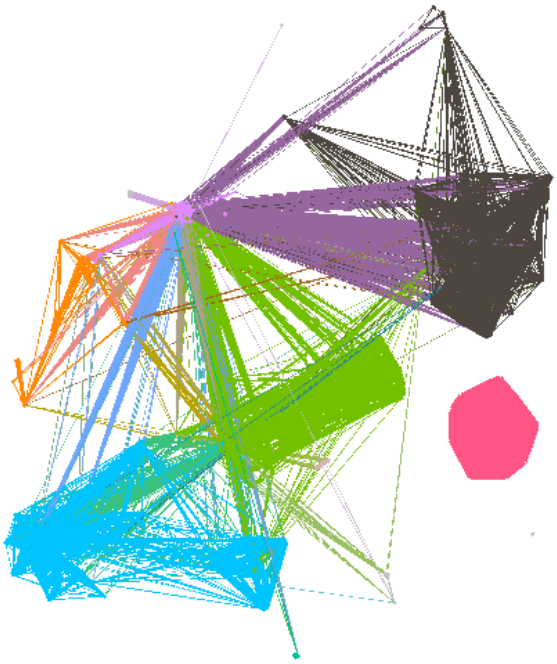
Εικόνα 7Κοινότητες χρηστών με βάση την ομοιότητα συνημιτόνου και του πεδίου των retweets



Εικόνα 8Κοινότητες χρηστών με βάση την ομοιότητα Jaccard και του πεδίου των retweets

Εύρεση κοινοτήτων χρηστών χρησιμοποιώντας την συνολική ομοιότητα

Εικόνα 9Κοινότητες χρηστών με βάση την ομοιότητα συνημιτόνου και την συνολική ομοιότητα



Εικόνα 10Κοινότητες χρηστών με βάση την ομοιότητα Jaccard και την συνολική ομοιότητα

Σύγκριση Δομών Κοινοτήτων

Μέσα από την σύγκριση των δομών κοινοτήτων, στόχος είναι η παρατήρηση της ομοιότητας ή της διαφοράς που παρουσιάζουν οι ομαδοποιήσεις που έγιναν με τα διαφορετικά μέτρα ομοιότητας. Για το σκοπό εξάγαμε σε μορφή CSV τις αναθέσεις χρηστών σε κοινότητες (μετρική modularity στο Gephi) και προβήκαμε στον υπολογισμό της μετρικής NMI (Normalized Mutual Information) μέσα από κώδικα σε Java.

Σύμφωνα με τους Danon et al. [9], δοθέντων δύο γράφων A και B, με πλήθος κοινοτήτων c_A και c_B αντίστοιχα, η μετρική NMI ορίζεται ως:

$$I(A,B) = \frac{-2 * \sum_{i=1}^{c_A} \sum_{j=1}^{c_B} N_{ij} * \log\left(\frac{N_{ij}N}{N_i N_j}\right)}{\sum_{i=1}^{c_A} N_i \log\left(\frac{N_i}{N}\right) + \sum_{j=1}^{c_A} N_j \log\left(\frac{N_j}{N}\right)}$$

όπου:

4. το N_i αποτελεί το πλήθος των κόμβων στην κοινότητα i,
5. το N_j αποτελεί το πλήθος των κόμβων στην κοινότητα j,
6. το N_{ij} αποτελεί το πλήθος των κόμβων στην ενοποιημένη κοινότητα i/j.

Αν οι εκάστοτε κοινότητες χρηστών είναι ταυτόσημες, τότε η μετρική NMI λαμβάνει την μέγιστη τιμής της, 1. Αντίθετα, αν οι κοινότητες διαφέρουν εντελώς μεταξύ τους, τότε η μετρική NMI λαμβάνει την ελάχιστη τιμή της, 0.

Η σύγκριση των δομών κοινοτήτων έγινε με βάση το διαφορετικό πεδίο – πώς δηλαδή διαφέρουν οι κοινότητες χρηστών που παρατηρούνται με βάση το μέτρο ομοιότητας Cosine και Jaccard ανά διαφορετικό πεδίο των tweets. Σύμφωνα, λοιπόν, με τους υπολογισμούς, προέκυψαν οι ακόλουθες μετρικές NMI:

FIELD	**NMI**
HASHTAGS	0.960191
URLS	0.284636
MENTIONS	0.985591
RETWEETS	1.0
OVERALL	0.957625

Αναφορές

[1] Twitter Streaming API https://dev.twitter.com/streaming/overview

[2] Spark Streaming http://spark.apache.org/streaming/

[3] MongoDB https://www.mongodb.com/

[4] PostgreSQL https://www.postgresql.org/

[5] Gephi https://gephi.org

[6] Okeanos http://okeanos.grnet.gr

[7] Sidorov, Grigori; Gelbukh, Alexander; Gómez-Adorno, Helena; Pinto, David. "Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Model". Computación y Sistemas. 18 (3): 491–504. doi:10.13053/CyS-18-3-2043.

[8] Jaccard, Paul (1901), "Étude comparative de la distribution florale dans une portion des Alpes et des Jura", Bulletin de la Société Vaudoise des Sciences Naturelles, 37: 547–579

[9] Danon, L.; Diaz-Guilera, A.; Duch, J.; and Arenas, A. 2005. Comparing community structure identification. Journal of Statistical Mechanics: Theory and Experiment P09008(0505245).