NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

MASTER OF SCIENCE
IN INFORMATION AND DATA MANAGEMENT

M.Sc. Thesis

Finding Topic-specific Trends and Influential Users in Social Networks

Christos D. Daskalakis

Supervisors: **Tsalgatidou Afroditi,** Associate Professor
Koutrouli Eleni, Post Doc Researcher

ATHENS

JULY 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΣΤΗ ΔΙΑΧΕΙΡΙΣΗ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΔΕΔΟΜΕΝΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΕΥΡΕΣΗ ΤΩΝ ΤΑΣΕΩΝ ΓΙΑ ΣΥΓΚΕΚΡΙΜΕΝΑ ΘΕΜΑΤΑ ΚΑΙ ΑΝΑΓΝΩΡΙΣΗ ΤΩΝ ΧΡΗΣΤΩΝ ΜΕ ΤΗ ΜΕΓΑΛΥΤΕΡΗ ΕΠΙΡΡΟΗ ΣΤΑ ΚΟΙΝΩΝΙΚΑ ΔΙΚΤΥΑ

Χρήστος Δ. Δασκαλάκης

Επιβλέπουσες: **Τσαλγατίδου Αφροδίτη,** Αναπληρώτρια Καθηγήτρια
Κουτρούλη Ελένη, Μεταδιδακτορική Ερευνήτρια

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2020

M.Sc. Thesis

Finding Topic-specific Trends and Influential Users in Social Networks

Christos D. Daskalakis
A.M.: M1440

JULY 2020

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εύρεση των τάσεων για συγκεκριμένα θέματα και αναγνώριση των χρηστών με τη μεγαλύτερη επιρροή στα κοινωνικά δίκτυα

Χρήστος Δ. Δασκαλάκης
Α.Μ.: Μ1440

Ιούλιος 2020

ABSTRACT

Social networks (SNs) have become an integral part of contemporary life, as they are increasingly used as a basic means for communication with friends, sharing of opinions and staying up to date with news and current events. The general increase in the usage and popularity of social media has led to an explosion of available data, which creates opportunities for various kinds of utilization, such as predicting, finding or even creating trends.

In this thesis, we first begin with analyzing the current work regarding the recommendation systems and their value on today's social networks as well as the internet. We review the literature on reputation systems, their importance and the use cases that can be found on modern online applications. Our goal is to combine the two aforementioned systems in order to identify the most influential users, not only based on their followership, but also in their respective fields. An influencer is someone who is known and trusted by a specific audience on specific topics. We try to distinguish between celebrities, and social media users that actually are respected in their fields and that other users trust and follow actively (user engagement).

We then extend our system, in order to also identify the most influential URLs, using the same metrics as before.

Our approach is based on the functionality of hashtags, which we use as topic indicators for posts, and on the assumption that a specific topic is represented by multiple hashtags. We present a neighborhood-based recommender system, which we have implemented using collaborative filtering algorithms in order to (a) identify hashtags, URLs and users related with a specific topic, and (b) combine them with SN-based metrics in order to address the aforementioned questions in Twitter. The recommender system is built on top of Apache Spark framework in order to achieve optimal scaling and efficiency. For the verification of our system we have used data sets mined from Twitter and tested the extracted results for influential users and URLs concerning specific topics in comparison with the influence scores produced by a state-of-the-art influence estimation tool for SNs. Finally, we present and discuss the results regarding two distinct topics and also discuss the offered and potential utility of our system.

SUBJECT AREA: Recommender Systems, Influencer Systems, Social Networks

KEYWORDS: influence, social networks, collaborative filtering, influence estimation

ΠΕΡΙΛΗΨΗ

Τα κοινωνικά δίκτυα έχουν γίνει αναπόσπαστο κομμάτι της σύγχρονης ζωής. Κάθε μέρα όλο και περισσότεροι άνθρωποι χρησιμοποιούν αυτά τα δίκτυα για να επικοινωνούν με τους φίλους τους, να μοιράζονται τις απόψεις τους και να μένουν ενημερωμένοι για νέα και τρέχοντα γεγονότα. Η γενική αύξηση της χρήσης και της δημοτικότητας των κοινωνικών μέσων ενημέρωσης οδήγησε σε μια έκρηξη των διαθέσιμων δεδομένων, γεγονός το οποίο δημιούγησε νέες ευκαιρίες για διάφορα είδη εκμετάλλευσης, όπως η πρόβλεψη, η εύρεση αλλά και η δημιουργία τάσεων.

Στην παρούσα διπλωματική εργασία, ξεκινάμε αναλύοντας την βιβλιογραφία που υπάρχει για τα συστήματα συστάσεων, καθώς και την αξία τους, για τα μέσα κοινωνικής δικτύωσης που υπάρχουν σήμερα, αλλά και γενικότερα για το ίντερνετ. Μετέπειτα αναλύουμε την συνεισφορά των συστημάτων εύρεσης επιρροής, της σημασίας τους αλλά και των χρήσεων τους στην μοντέρνες εφαρμογές διαδικτύου.

Στόχος μας είναι να συνδυάσουμε τα δύο προαναφερθέντα συστήματα, προκειμένου να εντοπίσουμε τους πιο σημαντικούς χρήστες, όχι μόνο με βάση την δημότικότητα τους, αλλά και στους τομείς που ασκούν την μεγαλύτερη επιρροή. Ένας χρήστης που ασκεί μεγάλη επιρροή, είναι κάποιος που είναι γνωστός και αξιόπιστος από ένα συγκεκριμένο κοινό και σε συγκεκριμένα θέματα. Με την προσέγγισή μας προσπαθούμε να κάνουμε διάκριση μεταξύ διασημοτήτων και χρηστών κοινωνικών μέσων που πραγματικά γίνονται σεβαστά στους τομείς τους, και άλλοι χρήστες εμπιστεύονται και ακολουθούν ενεργά (αφοσίωση χρηστών).

Στη συνέχεια επεκτείνουμε το σύστημά μας, προκειμένου να εντοπίσουμε και τις πιο σημαντικές διευθύνσεις URL, χρησιμοποιώντας τις ίδιες μετρηκές με πριν.

Η προσέγγισή μας για την απάντηση στις παραπάνω ερωτήσεις βασίζεται στη λειτουργικότητα των hashtags τα οποία θεωρούμε ότι αντιπροσωπεύουν κάποιο θέμα σε κάθε δημοσίευση και στην υπόθεση ότι υπάρχουν πολλαπλά hashtags που αντιπροσωπεύουν ένα συγκεκριμένο θέμα. Σε αυτή τη διπλωματική, παρουσιάζουμε ένα σύστημα συστάσεων, το οποίο υλοποιούμε χρησιμοποιώντας Collaborative Filtering αλγορίθμους για (α) τον εντοπισμό των hashtags, των Urls και των χρηστών που σχετίζονται με ένα συγκεκριμένο θέμα, (β) συνδυασμό διαφόρων μετρικών επιλεγμένων από το κοινωνικό δίκτυο μαζί τους και (γ) να αντιμετωπίσει έτσι τις προαναφερθείσες ερωτήσεις στο Twitter. Το σύστημά υλοποιήθηκε με την χρήση του Apache Spark προκειμένου να επιτευχθεί η βέλτιστη κλιμάκωση και να είναι σε θέση να επεξεργάζεται αποτελεσματικά μεγάλους όγκους δεδομένων.Για την επαλήθευση των αποτελεσμάτων του συστήματός μας χρησιμοποιήσαμε δεδομένα τα οποία αντλήσαμε από το Twitter. Τελικά συγκρίνουμε τα αποτελέσματά μας με τα αποτελέσματα επιρροής που παράγονται από δύο εργαλεία εκτίμησης της επιρροής κοινωνικών δικτύων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Συστήματα Συστάσεων, Συστήματα εύρεσης κόμβων με υψηλή επιρροή

ΛΕΞΕΙ ΚΛΕΙΔΙΑ: επιρροή, κοινωνικά δίκτυα, συνεργατικό φιλτράρισμα, εκτίμηση επιρροής

ΕΥΧΑΡΙΣΤΙΕΣ

Για την διεκπεραίωση της παρούσας ερευνητικής εργασίας, θα ήθελα να ευχαριστήσω την επιβλέπουσα καθηγήτρια κ. Αφροδίτη Τσαλγατίδου για την συνεργασία και την πολύτιμη συμβολή της στην ολοκλήρωση της, καθώς και την κ. Κουτρούλη Ελένη για την καθοριστική συνεισφορά και βοήθειά της καθ' όλη την διάρκεια εκπόνησης της εργασίας μου.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PROLOGUE

This thesis was developed for the master program of the Department of Informatics and Telecommunications of the national and Kapodistrian University of Athens, in the area of Information and Data Management. In this project, we present a neighborhood-based recommender system, which is implemented using collaborative filtering algorithms. We then combine the results from the collaborative filtering mechanisms with influence estimation techniques in order to find topic-specific trends both for content and for users.

In this thesis project we use Apache spark,s an open-source cluster computing framework which is used in order to achieve optimal scaling and to process large volumes efficiently.

1. INTRODUCTION

E-communities necessitate mechanisms for the identification of credible entities which can be trusted and used in a particular context. Various reputation systems have been proposed to effectively address this need [20], based on the evaluation of individual transactions. In social networks (SNs), where there is an abundance of information, rich social activity of many people and dynamic relationships and interactions of various forms, apart from the need to find credible entities, new requirements and new possibilities arise, which are related to the identification of influential entities, i.e. entities which attract the interest of users and can provoke actions [21]. A vast amount of research works have focused in the exploration of the concept of influence in social networks, its estimation and its use [1], [2], [3], [15]. These works are usually based on the various relationships and social actions of entities and focus on different aspects such as identification of influential entities and content [1], [15], influence propagation [28], influence maximization [7], combination of influence and trust [26]. Collaborative Filtering (CF) mechanisms have also been widely studied and used to identify similarities and to produce recommendations in SN-based applications [9], [16], [27].

SN applications include also the useful “hashtag” functionality, i.e. the possibility to tag content using hashtags, which is a way of mapping content to specific topics. This functionality, when combined with the vast amount of social network activity information, creates the opportunity to explore influence in a more specialized context. Useful examples of specialized influence estimation include finding influential news and influential users regarding the specific topic. In this thesis, we present our work towards answering the following questions: (a) Which are the most influential - popular internet publications posted in SNs for a specific topic? (b) Which members of SNs are experts or influential regarding a specific topic? Answers to these questions are vital in various areas such as marketing, politics, social media analysis, and generally in all fields, which need to quickly understand and respond to current trends.

Our approach towards answering the aforementioned questions combines influence estimation techniques and collaborative filtering mechanisms. More specifically, it is based on (a) the hashtag functionality and the assumption that a topic is represented by one or more hashtags, (b) collaborative filtering techniques for finding similar hashtags based on their common usage and the links assigned to them, and (c) analysis of social network-based actions. The contribution of this work is thus a solution for finding topic-specific trends both for content and for users: collaborative filtering is used for identifying a set of similar hashtags, which represent a topic, which are then used for filtering user activity in order to find the topic-specific influential users and content. This solution is, to the best of our knowledge, novel with respect to related works which examine influence from different perspectives.

Taking as an assumption that a specific hashtag represents a specific topic and as modern social media marketing dictates, assuming that SN users always use hashtags that represent the same topic using different representations, we are going to find the most influential users regarding a specific topic as well as the most influential link regarding that specific topic.

For the purposes of this thesis we developed a system that takes as input posts with a specific format. Our findings in the next chapters are focused on tweeter, but can be used for any SN, given that the posts that will be given as input to our system will be transformed in the common data model that our system supports.

In the following section we present related work which focuses on influence estimation in SNs. This is followed by an overview of our approach and a description of its steps. In the fourth section we present the implemented system and its evaluation using various scenarios and in relation to a benchmark influence estimation tool and we discuss the produced results. Our conclusive remarks follow in the last section.

.

2. RECOMMENDER SYSTEMS

2.1 Overview

With the recent rise of social media as well as the smart phones and tablets, the amount of information available on the Internet has grown exponentially. Useful information for users is getting harder to be found. So, the need for a technology that recommends interesting data to the user is necessary.

Recommender systems are used in order to produce a list of recommendations through collaborative filtering techniques or content-based filtering approaches in order to help a user discover products or content by predicting the user’s rating of each item and then present the user with the produced list. With the term item we are referring to products in an online shop, to posts on Facebook, to series on portals like Netflix. The recommendations made by all these sites are specific for a given user or for a group of users that have similar tastes.

CF techniques use a database of preferences for items by users to predict additional topics or products a new user might like. In a typical CF scenario, there is a list of m users and a list of n items, and each user, has a sub-list of these items that he/she has already rated. The ratings can be indications such as 1-5 stars ratings, or indirect interest like purchases, click-throughs etc.

There are many challenges for collaborative filtering tasks. CF algorithms are required to have the ability to deal with highly sparse data, to scale in order to satisfy the continuously increasing number of users and items, making recommendations in acceptable by the systems times.

2.2 Collaborative filtering

Collaborative filtering is a technique commonly used to build personalized recommendations on the web. Some popular websites that make use of the collaborative filtering technology include Amazon, Netflix, iTunes, IMDB and many others. One key reason why we need a recommender system in modern society is that people have too many options to use from due to the prevalence of Internet.

Collaborative filtering (CF) can be categorized into two main methods as user-based collaborative filtering (memory-based) and item-based collaborative filtering (model-based):

- Memory Based: This method makes use of user rating information to calculate the likeness between the users or items. The first approaches' basic idea is to identify similar users in a system and propose items that these users would find interesting and of use to them based on the preferences of other similar users. The above systems are call User-Based recommendation systems. In contrast, the second approach (item-item filtering) will take an item, in order to find the most similar items, based on characteristics and propose them to users. This kind of system can be found in a lot of online e-shops like Amazon. ("The users that bought this item, also bought…").

- Model Based: Models are created by using data mining, and the system takes advantage of Machine Learning algorithms in order to predict users ratings of unrated items.

2.2.1 Hybrid Techniques

All the above-mentioned types have strengths and weaknesses. In order to overcome the weaknesses, hybrid techniques, that combine the aforementioned types of models have been developed in order to meet the requirements of the different scenarios and to increase performance. In general, the hybrid techniques are especially used to solve the new user problem [30].

- Weighted recommender

  The score of different recommender systems are combined into one single individual recommendation. Before combining them, the scores are weighted according to their influence on a specific item.

2.3 Collaborative Filtering Process

In a fundamental scenario, collaborative filtering processing can be mainly divided into three steps:

- Collecting user ratings data matrix
- Selecting similar neighbors by measuring the rating similarity

- Generating prediction for recommended items

2.3.1 User Ratings

In recommendation systems the input data are usually ratings by users for specific items as a matrix m x n as shown in figure 1. Symbol m is the total number of users and n is the total number of items. $R_{m,n}$ is the rating of item n from user m.

Item / User	I1	I2	I3	...	In
U1	$R_{1,1}$	$R_{1,2}$	$R_{1,3}$	$\dots$	$R_{1,n}$
U2	$R_{2,1}$	$R_{2,2}$	$R_{2,3}$	$\dots$	$R_{2,n}$
U3	$R_{3,1}$	$R_{3,2}$	$R_{3,3}$	$\dots$	$R_{3,n}$
...	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
Um	$R_{m,1}$	$R_{m,2}$	$R_{m,3}$	$\dots$	$R_{m,n}$

Figure 1 User-Item ratings matrix

2.3.2 The formation of neighbors

The collaborative filtering approaches use statistical techniques to analyze the similarity between users in order to create a set of users that have common interests called neighbors. A set of similarity measures is a metric of relevance between two vectors. User-based similarity is to compute the relevance between users as the values of two vectors. In user based collaborative filtering, after the similarity is calculated, it is used in building neighborhoods of the current target user.

In contrast, in item based collaborative filtering there are no neighborhoods formed after the similarity scores are calculated. This is because the similarity between co-rated items is computed only as the value of two vectors.

2.3.2.1 Cosine similarity

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. If is often used to measure document similarity in text analysis. In our work we use it to find the most similar topics using the hashtag functionality. We also use the Euclidean distance similarity that we are going to explain in the next paragraph.

2.3.2.2 Euclidean distance

The basis of many measures of similarity and dissimilarity is Euclidean distance. The distance between vectors X and Y is defined as follows:

$$d(x,y) = \sqrt{\sum_{i}^{n} (x_i - y_i)^2}$$

Euclidean distance is basically the square root of the sum of squared differences between corresponding elements of the two vectors. This measure is appropriate only for data measured on the same scale. It is mostly used to compare profiles of respondents across variables.

3. REPUTATION AND INFLUENCE ESTIMATION SYSTEMS

3.1 Influence in SNs: An Implicit Reputation Measure

Reputation systems are systems that allow the users of an online community to rate each other in order to build trust through the estimation of the reputation of users or content. Reputation systems may also use implicit evaluations of user behavior, such as social relationships or similarity between users. They were essential tools long before the internet existed, with the word of mouth marketing being the main tool of such systems. In the last years, a new class of reputation systems has emerged to support social networks, which have changed the way users exchange services, opinions and make a new kind of trust relationships with each other. Online social networks connect users and facilitate content sharing in various ways and with various objectives, such as content sharing, blogging etc. The concept of reputation has new and distinct meaning, such as the influence that a user has developed towards other users. Effective reputation systems must carefully choose which aspects of user behavior should be tracked and reported.

In this thesis, we propose such a reputation system in order to find the most influential users, content, based on specific topics using the hashtag functionality that is nowadays part of almost all the online social networks.

3.2 Estimating Influence in SNs

Various works have focused on estimating influence of entities and content in social networks. Influence is dealt with in various ways: e.g. as an indirect reputation concept [15], [1], i.e. an indication of how much trust or popularity can be assigned to an entity or content based on indirect information rather than on direct ratings, or as an indication of action propagation [11], or from a social analysis perspective, [26], etc.. The various approaches use data related with the social activity in the SN, i.e. the actions of users towards other users or content in the SN. Most specifically, these approaches use algorithms which combine (a) entity-centered characteristics related with social actions, e.g. the number of likes of a post or the number of followers of a user [6] [11] and (b) social action-related characteristics of entity pairs, e.g. the number of likes user A assigns to posts of user B [26]. Trust relationships between two entities are also incorporated in the latter case, while a usual representation of a SN in the context of influence estimation is a graph where the nodes represent individual entities and the edges represent the links between two entities, accompanied with one or more weights (each one for different kinds or relationships).

Along with the different kinds of information used, related works differentiate according to: (a) influence estimation technique (b) requirements stemming from the kind of social network where influence is estimated.

Various influence estimation techniques are found in the literature, such as probabilistic [2], deterministic [15], graph-based [26], [6] and machine-learning-based (for influence prediction) [11].

According to the specific kind of social network, different requirements for influence estimation occur. For example, in microblogging social networks, such as Twitter [25], influence is mostly related with a number of factors, such as recognition and preference [1] ,[15] which are attributed to social network activity of users (numbers of shares, likes, followers, the followers social activity, etc.). In other works, such as in review social networks, e.g. Epinions [10], a combination of social activity information with trust relationships is used [26].

In Section 3.2.1, we briefly present some works related with influence estimation in SNs with a focus on the influence estimation technique and the data they use.

3.2.1 Literature on Influence estimation in SNs

- Agarwal et al. [1] deal with estimating the influence of bloggers in individual blogs. Four factors are considered as vital for defining influence: recognition, activity generation, novelty and eloquence. These properties are defined according to specific post characteristics and the social activities of bloggers, and are then combined for assessing the user’s influence.
- Anger et al. [3] measure influence of both users and content in Twitter. They take into consideration various Twitter statistics, such as the numbers of followers, tweets, retweets and comments, and they estimate separately two measures, one based on the content and one based on the action logs. Similarly, to this, the work in [15] presents an influence estimation system both for Twitter hashtags and users based on various social activity-based data. Further performance indicators for Twitter are presented in [3] ,including Klout [14], a widely accepted influence estimation tool for social networks.

- A different approach for finding the most important URLs regarding a specific topic in Twitter is proposed by Yazdanfar et al.[27], who reason about the importance of url recommendation in Twitter and implement such recommendations using collaborative filtering techniques and a three-dimensional matrix of users, urlurls and hashtags.
- Ahmed et al. [2] integrate the concept of trust in their approach for estimating influence probabilities. Their suggested algorithm discovers the influential nodes based on trust relationships and action logs of users. Varlamis et al. [26] integrate also trust relationships in their influence estimation mechanism, which uses both social network analysis metrics and collaborative rating scores, where the latter take into account both the direct and the indirect relationships and actions between two users.
- In [11] various influence models are constructed for a number of different time models and various algorithms used in the literature are analyzed and discussed. Bento [6] implements various social network analysis algorithms for finding influential nodes in location-based SNs and in static SNs.

From the perspective of influence as an indirect reputation concept, we have used a taxonomy of indirect reputation systems [15], in order to present the specific characteristics of the aforementioned works in Table 1, with a focus on the information exploited and the influence estimation technique.

Our approach focuses on topic-specific influence, but unlike topic-specific recommendation systems for SNs, such as [6], it is not restricted to collaborative filtering techniques. Furthermore, it is not restricted SN activity–based influence estimation, which is adapted in [2], [1],[15]. It comprises rather a specialized influence estimation which combines the user activity characteristics responsible for influence estimation with collaborative filtering techniques for the topic-specific filtering of posts.

Table 1. Characteristics of Influence Estimation Systems

	"Social Networking Metrics"- Varlamis et al. [26]	"Measuring Influence On Twitter"- Anger et al. [3]	"Influential bloggers"- Agarwal et al. [1]	"Influential nodes from trust network"- Ahmed et al. [2]	"Learning influence probabilities"- [11]	"Link Recommender"- Yazdanfar et al.[27]
Description of influence estimation technique	(-) calculates score by user's position on the SN Graph (-) calculates user's opinion score by the user's reviews to articles and other users	Takes into consideration various Twitter statistcs that calculate different types of Influence and combines them in one model	Influence of bloggers based on the influence of one's posts according to - - -post characteristics and -social activities of bloggers, e.g. references to a post	Influence probabilities are estimated based on trust relationships and action logs.	Calculates influence of a user and his influence probability for static and continuous time model	Produces url recommendations based on specific topics
Target	User	User and Content	User	User	User	Content
Scope	Combination of Personalized and Global	Global	Global	Global	Global	Global
Information Used	Information Regarding User position on SN Graph and Metrics about users each other opinions	Number of followers, tweets, retweets, mentions, comments	Various information regarding social activity	Trust relatinships and action logs	Action log about user's interactions and graph of the SN	Information related to the use of hashtags
Key Computational Comonents	Sums all SN Graph position information with the opinion scores and each value weighted	Adds conversation-oriented and content-oriented values separately and divides the result by two (2)	Statistic functions	Influence probabilities based on action log and trust relationships Suggested algorithm discovers the influential nodes	Probability estimation using	Collaborative-filtering techniques
Display Method	Ranked list of users	Ranked list of users and content	Ranked list of users	Ranked list of users	Influence Matrix, Table of activated (influenced) users	Ranked list of urls

4. PROPOSED SYSTEM

4.1 Estimating Influence on a Specific Topic

The goal of the proposed approach is to estimate influence of users and urls regarding a specific topic, and to find the most influential ones among them. The idea is that we first choose a hashtag which is representative of the topic of interest and then find a set of hashtags which are similar to the initial hashtag. We then aggregate social network metrics for the tweets which have used these hashtags in order to estimate influence scores for the users which have posted these tweets and for the urls which have been used in them.

For the purposes of this thesis we focused on micro blogging systems like Twitter [25]; however, the proposed approach can be generalized due to the fact that its elements (e.g. hashtags, numbers of likes and followers) are common to most social networks.

Here is a step-by-step description of the approach we follow:

- Step 1: Given a specific hashtag hi, we first find the N-top similar hashtags based on collaborative filtering techniques which take into consideration the level of usage of hashtags by users and the usage of common urls together with hashtags, as explained in section 4.1. We define H as the set containing hi and the most similar hashtags to hi.
- Step 2: We collect the sets of tweets, users and urls which have used at least one hashtag belonging to H, as analytically presented in section 4.2.
- Step 3: Based on the above sets, we find the most influential users. The criteria for estimating a user’s influence are based on social activity-based metrics related to tweets which contain the specific url. This step is presented in section 4.3.
- Step 4: In a similar way, we use the above sets to find the most influential urls, using various social activity-based criteria, as described in section 4.4.

4.2 Finding Similar Hashtags for Topic Representation

In order to identify a set of hashtags which represent a topic, we use an initial hashtag “h” and try to find hashtags which are similar to h, using two criteria for assessing similarity: (a) the number of common links that two distinct hashtags have (if two hashtags have the same number of references to a link, this link is related to the same level to these hashtags) and (b) the level of their usage by users who have used them in common (if two users have used them with similar frequency this means that these hashtags are of the same level of interest for the users, and are considered similar in this context). We thus define two similarity measures for hashtags according to the two criteria and combine them in one. Specifically, we use the similarity measures (1) and (2) that appear below, in order to estimate the Euclidean distance of two hashtags regarding the two criteria.

$$sim_{euclidean}(h_i, h_j)_{url} = 1 \Big/ \left(1 + \sqrt{\sum_{l=1}^{L} \left(r_{h_{i,l}} - r_{h_{i,l}}\right)^2}\right) \qquad (1)$$

where

- h_i, h_j are two distinct hashtags,
- L is the set of the links (urls) which have been used in at least one tweet of each of the hashtags h_i, h_j
- l is a url belonging to L,
- $r_{hi,l}$, $r_{hj,l}$ are the numbers of tweets which have used the link l and have also used the hashtag h_i and h_j, and
- $sim_{euclidean}(h_i, h_j)_{url}$ is the similarity of hashtags h_i, h_j regarding their usage of common links.

$$sim_{euclidean}(h_i, h_j)_{user} = 1 \Big/ \left(1 + \sqrt{\sum_{u=1}^{U} \left(r_{h_{i,u}} - r_{h_{i,u}}\right)^2}\right) \qquad (2)$$

where

- hi, hj are two distinct hashtags,
- U is the set of the users which have used the two hashtags in their tweets,
- u is a user belonging in U,
- $r_{hi,u}$, $r_{hj,u}$ are the numbers of tweets of user u which have used the hastag h_i and h_j, and
- $sim_{euclidean}(h_i, h_j)_{user}$ is the similarity of hashtags h_i, h_j regarding their common usage by users.

We also use the cosine similarity measure presented in (3) to estimate the similarity between two hashtags according to the criteria of the level of interest and the number of commonly used urls (sim_{cosine} $(h_i,h_j)_{user}$, and sim_{cosine} $(h_i,h_j)_{urlr}$ accordingly):

$$sim_{cosine}(h_i, h_j)_{url/user} = \frac{\sum_{k=1}^{n}(r_{ik})*(r_{jk})}{\sqrt{\left[\sum_{k=1}^{n}(r_{ik})^2\right]}*\sqrt{\left[\sum_{k=1}^{n}(r_{jk})^2\right]}} \qquad (3)$$

where

- $sim_{cosine}(h_i, h_j)_{url}$ is the cosine similarity of hashtags h_i, h_j regarding their common urls,
- $sim_{cosine}(h_i, h_j)_{user}$ is the cosine similarity of hashtags h_i, h_j regarding their common usage by users,
- h_i, h_j are two distinct hashtags,
- n is the number of users which have both used the two hashtags (when $sim_{cosine}(h_i, h_j)_{user}$ is estimated) or the number of the common urls used by the two hashtags (when $sim_{cosine}(h_i, h_j)_{url}$ is estimated),
- k is a user (when $sim_{cosine}(h_i, h_j)_{user}$ is estimated) or a url (when $sim_{cosine}(h_i, h_j)_{url}$ is estimated),
- r_{ik}, r_{jk} are the numbers of times a user k has used the hashtags h_i, h_j respectivley (when $sim_{cosine}(h_i, h_j)_{user}$ is estimated) or the numbers of times a url k has been used in hashtags h_i and h_j respectively (when $sim_{cosine}(h_i, h_j)_{url}$ is estimated)

We use either one of the above similarity measures (Cosine or Euclidean distance-based similarity), or average measures to define final similarity metrics for user-based similarity ($sim(h_i,h_j)_{user}$) and url-based similarity ($sim(h_i,h_j)_{url}$). We then combine the two similarity measures using a weighted average to estimate the similarity between two hashtags.

$$sim(h_i, h_j) = w_{simuser} * sim(h_i, h_j)_{user} + w_{simurl} * sim(h_i, h_j)_{url} \quad (4)$$

where

- $w_{simuser}$, w_{simurl} are the weights we use for the two kinds of similarity, and
- $w_{simuser} + w_{simurl}$=1

We thus find a list of top-N hashtags which have the highest similarity with the original hashtag hi. We define as H, the set which contains hi and the N hashtags of this list. The original set of hashtags which are examined for the extraction of H can be obtained by various ways e.g. Twitter Streaming API [18], Twitter Rest API [24] Twitter widgets [23]. The selection of the original hashtag hi from the available hashtags can be done either based on personalized criteria, e.g. one can select a hashtag which she believes as representative of a topic, or by searching available hashtags with text similarity criteria.

4.3 Collection of Data

Having acquired the set **H** of hashtags, which represent a topic, we collect the following data, which are needed for finding the influential users and urls in the context of a specific topic:

1. The set $\mathbf{T_H}$ of all the tweets which have at least one hashtag belonging to the set **H**.
2. The set $\mathbf{U_H}$ of all users which have tweeted at least one tweet belonging to the set $\mathbf{T_H}$**,** i.e. users which have used hashtags belonging to H.
3. The set $\mathbf{L_H}$ of all urls which have been attributed to one or more tweets belonging to the set $\mathbf{T_H}$ (or equivalently to one or more hashtags belonging to the set **H**).

In the following sections, we describe the ways we use to extract lists of influential users and influential urls regarding a specific topic.

4.4 Finding Influential Users on a Specific Topic

For each one of the users belonging to in $\mathbf{U_H}$ we estimate her influence score regarding each hashtag belonging to H, based on the triple: *(weighted number of likes of related tweets, weighted number of retweets of related tweets, absolute number of user's followers)*. The triple is used to represent a number of criteria which we consider as important for determining influence. These criteria are presented in the rest of this section, along with the related metrics – formulae used for the estimation of a user's influence regarding a hashtag.

Estimating a User's Influence on a Specific Hashtag. The criteria for estimating a user's influence regarding a specific hashtag are described below, together with the metrics which represent them.

Adaptation: The total number of retweets a user has got for a specific topic (hashtag) shows the interest of other users to adapt or share the user's posts. We are interested in the adaptation level of u_i's tweets containing h_i, compared to the general level of adaptation that tweets containing h_i generate. We thus use the following adaptation metric:

$A(u_i, h_i)$ = the ratio of the number of retweets of the posts of a specific user u_i containing a specific hashtag h_i, to the total number of retweets which contain h_i. This metric shows the relative interest of users to share u_i's tweets compared to the total amount of interest that related tweets generate.

$$A(u_i, h_i) = \frac{number\ of\ retweets\ of\ u_i i's\ tweets\ which\ contain\ h_i}{number\ of\ retweets\ of\ all\ tweets\ containing\ h_i}$$

Preference: A user's influence can be measured by the number of her followers; the more friends a user has got, the more she is trusted / preferred.

$$P(u_i) = number\ of\ followers\ of\ u_i$$

Endorsement: (concerning a specific topic expressed by a hashtag h_i): In today's social networks every post of a user can be endorsed by other users. In Facebook you can endorse the post of a user by reacting to it (like, Wow, etc.), in Twitter you can declare you like it. The more users endorse a post, the more influence this post has over users. This gives us an insight of how valuable is the user's opinion on some topic. We are interested in the value of the user's opinion is on a topic, in relation to the value of other users' opinions on that topic. For the endorsement metric we have thus used the following formula:

$E(u_i, h_i)$= the ratio of the number of favorites that u_i's tweets containing a hashtag h_i have been assigned, to the total number of favorites assigned to tweets which contain

h_i.This metric shows the relative endorsement in user's tweets compared to the total endorsement for tweets containing h_i.

$$E(u_i, h_i) = \frac{number\ of\ favorites\ of\ u_i i's\ tweets\ containing\ \ h_i}{number\ of\ favorites\ of\ all\ tweets\ containing\ h_i}$$

We are using a weighted mean to estimate the influence score of a user u_i concerning a specific hashtag h_j, as a combination of the result scores of the above three influence factors:

$$InfUserHashtag(u_i, h_j) = w_A * A(u_i, h_j) + w_E * E(u_i, h_j) + w_P * P(u_i) \quad (5)$$

where

- w_A, w_P, w_F are the weights we assign to the factors described above, and
- $w_A + w_P + w_F = 1$.

The choise of the values of these weights should be done according to the importantce we want to give to each criterion. Machine learning methods can also be used to find the most appropriate values for the weights.

Estimating a User's Influence on a Topic. Having estimated the individual influence score of users regarding each (Top-N similar) hashtag of the hashtag set H which is representative of a topic, according to the previous section, we estimate the total influence score of every user using the following formula:

$$Inf(u_i) = \sum_{j=1}^{N} w_j * InfUserHashtag(u_i, h_j) \quad (6)$$

where

- $Inf(u_i)$ is the influence score of a user u_i,
- $InfUserHashtag(u_i, h_j)$ is the influence score of a user u_i concerning a specific hashtag h_j, where h_j belongs to the set of the top N similar hashtags H and has the jth order in similarity with the initial hashtatg), and

- w_j is the weight assigned to the score of each hashtag h_j,

We have adjusted the values of weighs w_j which we assign to the various InfUserHashtag scores, depending on the similarity of the specific hashtag to the initial hashtag according to (4). Specifically, considering that, $InfUserHashtag(u_i, h_1)$ $InfUserHashtag(u_i, h_2)$, …, $InfUserHashtag(u_i, h_N)$ are ordered according to their similarity with the initial hashtag, then the first score will refer to the initial hashtag itself (h_1) and its weight w_1 will be estimated according to (7) and will be used as reference for estimating the other weights, in a way that $w_{i+1} = (w_i/2)$.

$$w_1 = 1 \,/\, (1 + \textstyle\sum_{i=1}^{k} \frac{1}{2^i}) \qquad (7)$$

Where k: $2^k <= N * 2$ and $2^{k+1} > N * 2$Using this way of weight estimation, we achieve [1]and we give higher $2^{k+1} > N * 2$ weights to the influence scores of the most similar hashtags.

We finally extract the most influential users $\sum_{i=1}^{N} w_i \approx 1$ regarding the topic based on the estimated influence of users.

4.5 Finding Influential Urls on a Specific Topic

As explained in section 3.2, for each one of the hashtags belonging to the set $\mathbf{H}$ of similar hashtags, we find the links which have been used with them (set $\mathbf{L}_{\boldsymbol{H}}$) and the related tweets (set $\mathbf{T}_{\boldsymbol{H}}$). Then, for each link l of $\mathbf{L}_{\boldsymbol{H}}$ we find the subset $\mathbf{T}_{\boldsymbol{l}}$ of $\mathbf{T}$, which contain the tweets of $\mathbf{T}$ which have used this link. We also find the numbers of likes and retweets of the tweets belonging in $\mathbf{T}_{\boldsymbol{l}}$.

[1] When $2 * N = 2^k$ for an integer number k, then $\sum_{i=1}^{N} w_i = 1$.

We extract thus a list with recommended links, i.e. the ones with the highest influence score, which depends on the number of likes and retweets (of the relative tweets that contain them) and the number of tweets that contain these links, according to formula (5).

$$Inf(l) = w_{likes} * \left(\frac{TlNoLikes}{TlNoTweets}\right) + w_{retweets} * \left(\frac{TlNoRetweets}{TlNoTweets}\right) \qquad (8)$$

where

- $TlNoLikes$ is the number of likes on tweets belonging to $\mathbf{T}_l$,
- $TlNoRetweets$ is the number of retweets of tweets belonging to $\mathbf{T}_l$,
- $TlNoTweets$ is the number of tweets belonging to $\mathbf{T}_l$
- w_{likes}, $w_{retweets}$ are the weights of the numbers of likes and retweets respectively, related to the total number of topic-related tweets.

We consider these links as the most influential ones for a specific topic, since they are the ones which are mapped to the most related hashtags to the topic and also are attributed to the highest social activity.

5. IMPLEMENTATION OF THE PROPOSED SYSTEM

5.1 Overview

We have developed a system that implements the proposed approach, as a data analysis platform for Twitter, which features three core functionalities in Twitter:

- Finding the most similar topics (hashtags) using collaborative filtering techniques based on the posts gathered from a social network.
- Finding the most influential links in terms of popular internet publications, using the information extracted from the above step.
- Finding the most influential users for a specific toping, exploiting the set of similar topics from the first step.

In the following subsections we present the technology used, the collection of real social network data and the evaluation tests.

5.2 Technology and Datasets

For the implementation of this big data analysis platform we have created an application that leverages the possibilities provided by the Apache Spark framework [4], as well as the Java programming language. Apache Spark is used in order to achieve optimal scaling and the possibility to process large volumes of data faster and more efficiently.

For the purposes of this thesis we used the Twitter Rest API [24] in order to collect the last 1 to 3.300 tweets for different groups of Twitter users. The first group consisted of the most followed Twitter accounts. The second one had all the users that are considered the most influential ones for the Twitter platform according to [22] and the third one consisted of random Twitter users with a count of total users equal to 52. We used users from these three selected groups for our tests in order to have a wide range of users with different levels of popularity and influence in the social network. The tweets collected where filtered in order to contain at least one hashtag and were stored in the following format:

- UserId|TweetId|CreatedDate|Lang|text|FavCount|ShareCount|#|…|#|url|…|url|

For each one of the aforementioned users, we stored the following data concerning them:

- UserId|FollowersCount|FriendsCount|StatusesCount

We note that we intend to expand our experimental evaluation in larger datasets, leveraging the scalability possibilities offered by Apache Spark, in order (a) to extract results for influential urls and users in various scenarios involving different topics and user groups, and (b) to evaluate the performance of the system, e.g. running times in big data volumes.

5.3 Finding Topic-Specific Hashtags and Influential Users and Urls

From the 72.029 collected tweets we were able to extract 40.924 pairs of hashtags that (a) were used by at least two users in their tweets and (b) had at least one url in common. For each pair we were then able to define their similarity scores according to the formulae (1)-(4).

For estimating the total similarity between two hashtags, we chose the Cosine similarity metric, and used formula (4), with the following weight values: $w_{simuser}$=0,4 and w_{simurl}=0,6, considering that the similarity score of two hashtags regarding their common urls of greater importance than their similarity concerning their common level of usage by users. We note that the reason we chose to use the Cosine similarity metric for estimating both similarity components, is its suitability when we are interested in the cosine of the rating vectors [5], i.e. the similarity between their trends (in our case the trends of the url rating vectors and the trends of the user rating vectors).

Influential Users based on a Specific Topic. We have examined the influence scores of users for various topics and have compared our results with the scores provided by the user influence estimation tool InfluenceTracker [14] for the same users. We have chosen InfluenceTracker as a benchmark for our comparison, as it is included in the state of the art of influencer discovery and Twitter [19]. In the rest of this section we present preliminary evaluation results for the topics “marketing” and “bigdata” considering they are represented by the initial hashtags #marketing and #bigdata respectively. We have thus first extracted from our dataset the top most similar hashtags for #marketing and present them in Fig. 1, along with their similarity scores based on (4).

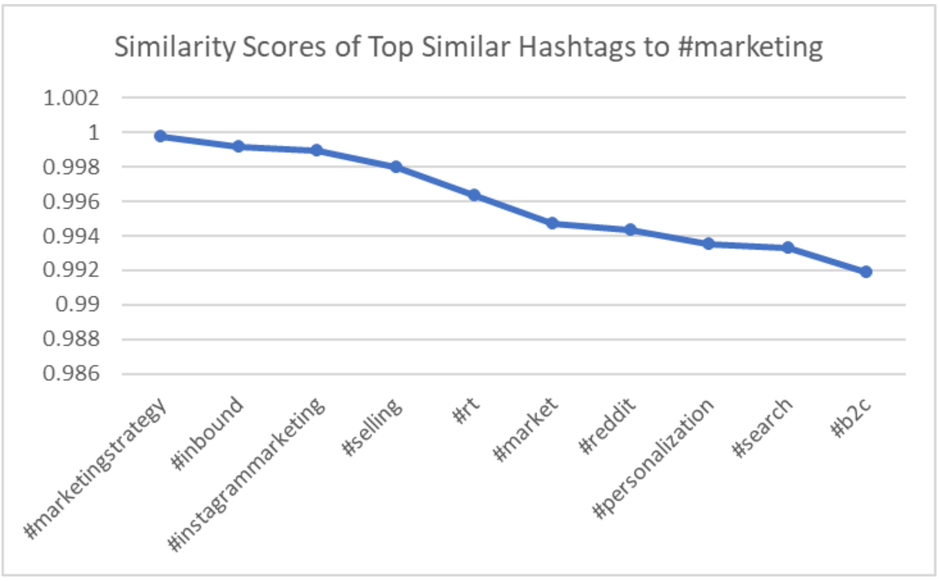


Figure 3. 10 Most similar hashtags with #Marketing

Fig. 3 presents the influence scores of the top ten most influential users regarding the topic "marketing", considering this topic is represented by the hashtag #marketing and its most similar hashtags presented above. For the same users we have also estimated the scores produced by InfluenceTracker [17] [13] and present them in Fig. 3. The InfluenceTracker score combines the numbers of a user's followers, followees and tweets. Its lowest value is "0", while the highest has no upper limit. The higher this value is, the more impact has an account on the social network [13].

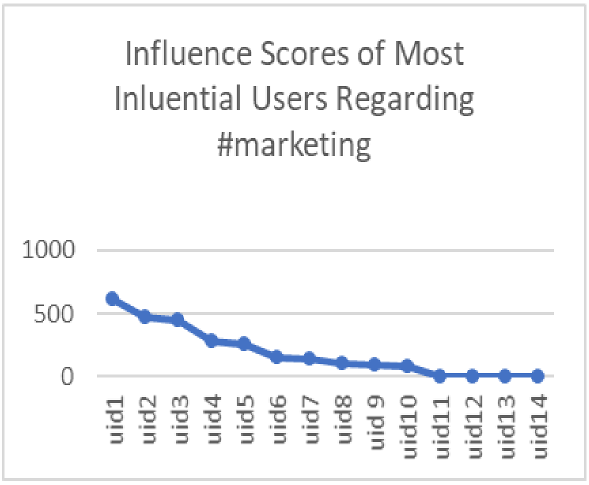


Figure 4. Influence scores of most influential users based on the topic #marketing

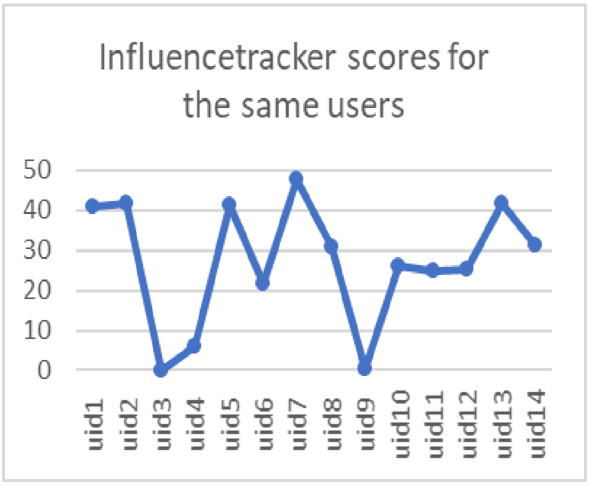
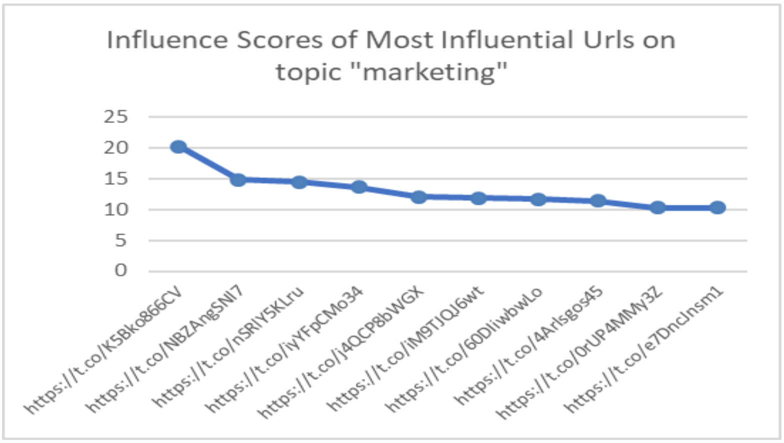


Figure 5. InflunceTracker scores for the same users

Figures 4 and 5 give different insights on the users, as InfluenceTracker takes into consideration all the tweets of a user, whereas the proposed systems is based on topic-related tweets. Differences are also due to the different time periods of the tweets that were used in the two metrics, as InfluenceTracker takes into consideration the last 100 tweets of a user, whereas our metric is based on the collected dataset.

Influential Links based on a Specific Topic. In order to extract the most influential links for a specific topic we have implemented the algorithm described in Section 3.4.



We have used as an example the topic “marketing” and the urls with the highest influence scores are shown in Fig.6. These urls redirect to articles written about marketing, and content marketing in social media. We have used as a second example the hashtag #bigdata assigning to the topic “big data”. We searched for hashtags similar to #bigdata and have identified the similar hashtags which are shown in Fig. 7. We have then estimated the influence scores of the users identified in the previous example as the most influential users regarding the topic “marketing”. The results of these users’ influence scores are presented in Fig. 8. It is evident that influence scores of the users differ according to the topic examined. We can see that for the “big data” topic the same users which were first examined for the topic “marketing” have different influencer scores. For example user uid1 that was the most influencing user in the marketing topic now has one of the lowest scores. Furthermore, users uid3, uid5 and uid7 have zero influence score in this specific topic, whereas they were considerably influencing regarding the topic “marketing”. We also found and present in Fig. 9 the most influential urls regarding “big data”. These urls redirect to articles related to the hashtags (#smartcities, #healthtech, etc.) that our platform identified as mostly related with the used topic.

Figure 6. Influence scores of most influential urls on the topic “marketing”

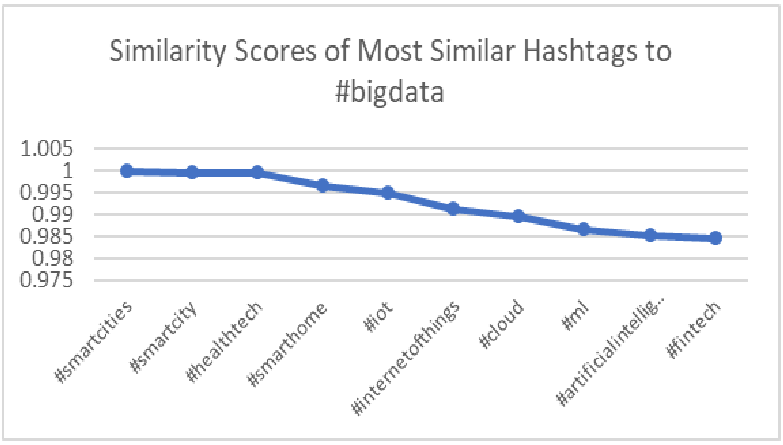


Figure 7. Most similar hashtags to #bigdata

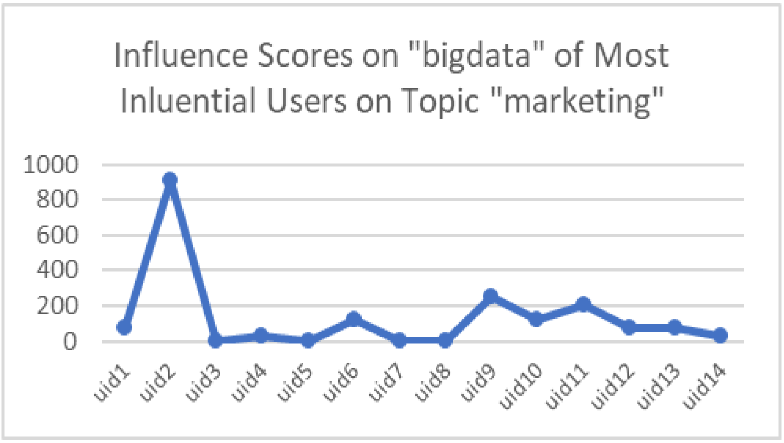


Figure 8. “Big data”-based Influence scores of most influential users on topic “marketing”

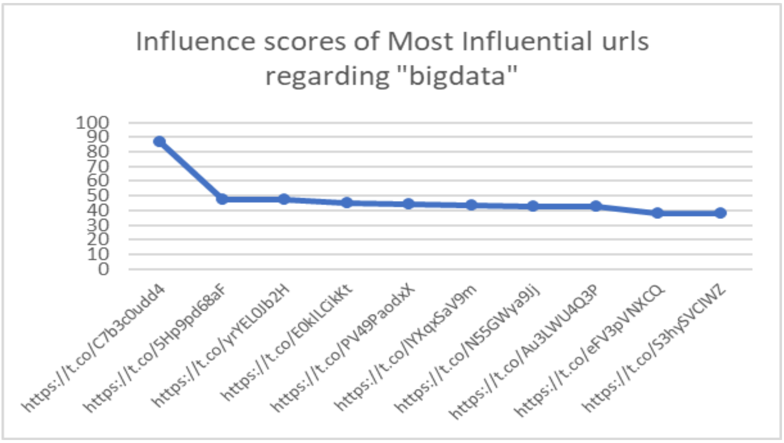


Figure 9. Influence scores of urls for topic “bigdata“

6. CONCLUSIONS

In this thesis we describe the design and implementation of a recommendation system for influential urls and influential users based on specific topics in Social Networks (SNs). For the purposes of our work, we use the hashtag functionality of SNs and, based on the assumption that the hashtags represent specific topics, we use collaborative techniques for identifying a pool of similar hashtags that correspond to a specific topic. We then use social activity based metrics for estimating the influence of urls and users by taking into consideration the identified hashtags, so as to achieve specialization on topics. For the implementation, the Apache Spark framework was used for achieving scalable searches and data processing. Results for specific topics based on the datasets we have extracted from Twitter were presented and analyzed. The benchmark influence estimation tools InfluenceTracker [13] [17] was used for a comparison of our results with the influence values this system estimates for the most influential users. Based on the preliminary evaluation of our results and the study of related work, we consider that our system comprises an innovative approach towards topic-specific influence estimation, and specifically towards revealing topic-specific trends in content (as the most influential urls) and topic-specific influencers. We note that further tests are included in our future plans for (a) examining more use cases with bigger data sets and evaluating the performance of the proposed system, (b) fine tuning the various weights and components of our system, and (c) comparing the results and the performance of the proposed system with other influence tools for SNs and with topic specialization in mind.

ABBREVIATIONS AND TERMS

SNs	Social Networks like Twitter, Facebook, Instagram etc.
URL	Uniform Resource Locator
Influencer	An individual who can drive another person, or a community of people, to take some kind of action
Twitter	Twitter is a 'microblogging' system that allows you to send and receive short posts called tweets
Hashtag (#)	A hashtag is simply a relevant word or series of characters preceded by the # symbol. Hashtags help categorize messages and can make it easier for other Twitter users to search for tweet
Apache Spark	Apache Spark is a unified analytics engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing.

APENDIX I

Code to subscribe to Twitter's live feed and save tweets to files that contain maximum 200000 tweets, using Twitter4j framework.

```
public class SubscribeToTwitterLiveFeedAndSaveTweets {


    public static int numOfFiles = 4;
//    public static int numOfFilesUser = 0;
    public static int numOfLines = 0;
//    public static int numOfLinesUser = 0;


    public static BufferedWriter outTweets;
//    public static BufferedWriter outUsers;



    public static void closeWriter(BufferedWriter out ){
        try {
            out.close();
        } catch (IOException ex) {

java.util.logging.Logger.getLogger(SubscribeToTwitterLiveFeedAndSaveTweets.class.g
etName()).log(Level.SEVERE, null, ex);
        }
    }
    public static BufferedWriter changeFiles(BufferedWriter out,int wh){
        try {
             if(wh == 0){
                numOfFiles++;
                if(numOfFiles == 20)
                    exit(0);
                out = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream("06042018/Tweets"+numOfFiles+".txt"),"UTF-8"));
            }
//          else{
//              numOfFilesUser++;
//              out = new BufferedWriter(new FileWriter("Users"+numOfFiles+".txt"));
//          }
        } catch (IOException ex) {

java.util.logging.Logger.getLogger(SubscribeToTwitterLiveFeedAndSaveTweets.class.g
etName()).log(Level.SEVERE, null, ex);
            return null;
        }
        return out;
    }
    public static void main(String[] args) throws TwitterException, IOException{


    StatusListener listener = new StatusListener(){
        public void onStatus(Status status) {


            if(numOfLines == 200000){
                numOfLines = 0;
                closeWriter(outTweets);
                outTweets = changeFiles(outTweets, 0);
            }
//          if(numOfLinesUser == 200000){
```

```
//              numOfLinesUser = 0;
//              closeWriter(outUsers);
//              outUsers = changeFiles(outUsers,1);
//          }

          if( status.getRetweetedStatus() == null){
              try {
                  if((status.getHashtagEntities() != null && status.getHashtagEntities().length
> 0)
                          && (status.getURLEntities() != null && status.getURLEntities().length
> 0)){
                          outTweets.write(status.getUser().getId()+"|"+status.getId()+
                                  "|"+status.getCreatedAt()+"|"+status.getLang()+
                                  "|"+status.getFavoriteCount()+"|"
                          +status.getRetweetCount()+"|");
                          for(HashtagEntity he :status.getHashtagEntities())
                              outTweets.write("#"+he.getText()+"|");
                          for(URLEntity ur :status.getURLEntities())
                              outTweets.write(ur.getURL()+"|");

                          if(numOfLines < 200000)
                              outTweets.newLine();
      //
outUsers.write(status.getUser().getId()+"|"+status.getUser().getName());
      //                  outUsers.newLine();
                          numOfLines++;
                  }
//                  numOfLinesUser++;
```

```
                } catch (IOException ex) {

java.util.logging.Logger.getLogger(SubscribeToTwitterLiveFeedAndSaveTweets.class.g
etName()).log(Level.SEVERE, null, ex);
                    return;
                }
            }



        }
        public void onDeletionNotice(StatusDeletionNotice statusDeletionNotice) {}
        public void onTrackLimitationNotice(int numberOfLimitedStatuses) {}
        public void onException(Exception ex) {
            ex.printStackTrace();
        }

        @Override
        public void onScrubGeo(long userId, long upToStatusId) {
            throw new UnsupportedOperationException("Not supported yet."); //To change
body of generated methods, choose Tools | Templates.
        }

        @Override
        public void onStallWarning(StallWarning warning) {
            throw new UnsupportedOperationException("Not supported yet."); //To change
body of generated methods, choose Tools | Templates.
        }
    };
    outTweets =  changeFiles(outTweets,0);
//    outUsers  = changeFiles(outUsers,1);


    TwitterStream twitterStream = new TwitterStreamFactory().getInstance();
    twitterStream.addListener(listener);
    // sample() method internally creates a thread which manipulates TwitterStream and
calls these adequate listener methods continuously.
    twitterStream.sample();


    }}
```

APPENDIX II

Code written for cosine similarity.

```
private static final PairFunction<Tuple2<HashTagHashTagPair,Iterable<RatingPair>>,HashTagHashTagPair,Float>  COSINE_SIMILARITY =
    new PairFunction<Tuple2<HashTagHashTagPair,Iterable<RatingPair>>,HashTagHashTagPair,Float>(){


        HashTagHashTagPair hhtp;
        float sum_mm = 0;
        float sum_nn = 0;
        float sum_mn = 0;
        float numerator = 0;
        float score ;
        float denominator = 0;
        @Override
         public Tuple2<HashTagHashTagPair, Float> call(Tuple2<HashTagHashTagPair, Iterable<RatingPair>> t) throws Exception {
            sum_mm = 0;
            sum_nn = 0;
            sum_mn = 0;
            numerator = 0;


            denominator = 0;
           t._2.forEach(x->{
             sum_mm += x.getRating1() * x.getRating1();
             sum_nn += x.getRating2() * x.getRating2();
```

```
                  sum_mn += x.getRating1() * x.getRating2();
                });
                numerator = sum_mn;
                score = 0;
                denominator = (float) ((float) Math.sqrt(sum_mm) * Math.sqrt(sum_nn));
                if(denominator != 0)
                    score = ((float)numerator/(float)denominator);

                return new Tuple2<HashTagHashTagPair, Float>(t._1,score);
            }
    };
```

Code written for Euclidean similarity.

```
private static final PairFunction<Tuple2<HashTagHashTagPair,Iterable<RatingPair>>,HashTagHashTagPair,Float>  EUCLIDEAN_SIMILARITY =
       new PairFunction<Tuple2<HashTagHashTagPair,Iterable<RatingPair>>,HashTagHashTagPair,Float>(){

           HashTagHashTagPair hhtp;

           float sum_euc = 0;
           float score ;
           float finalScore;
           @Override
              public Tuple2<HashTagHashTagPair, Float> call(Tuple2<HashTagHashTagPair, Iterable<RatingPair>> t) throws Exception {
                  sum_euc = 0;
```

```
                score=0 ;
                finalScore = 0;
               t._2.forEach(x->{

                sum_euc += (x.getRating1() - x.getRating2())*(x.getRating1() -
x.getRating2());
               });

               score = (float) Math.sqrt(sum_euc);
               finalScore = (float)1/(float)(1+score);

               return new Tuple2<HashTagHashTagPair, Float>(t._1,finalScore);
            }
    };
```

Code for weighted average.

```
private static final
PairFunction<Tuple2<HashTagHashTagPair,Tuple2<Float,Float>>,HashTagHashTagP
air,Float> FINAL_SCORE =
        new
PairFunction<Tuple2<HashTagHashTagPair,Tuple2<Float,Float>>,HashTagHashTagP
air,Float>(){

            float finalScore = 0;
          @Override
          public Tuple2<HashTagHashTagPair, Float>
call(Tuple2<HashTagHashTagPair, Tuple2<Float, Float>> t) throws Exception {
              finalScore = 0;
```

```
                finalScore = (float)((weightA*t._2._1) + (weightB*t._2._2)) / (weightA+weightB);

                return new Tuple2<HashTagHashTagPair,Float>(t._1,finalScore);

            }

                                                    };
```

Code to Identify influencers (example for #marketing)

```
String master = "local[*]";
          String hash = "#marketing";
            /*
             * Initializes a Spark context.
             */
            System.setProperty("hadoop.home.dir", "C:\\winutils\\");
            SparkConf conf = new SparkConf()
                .setAppName(Example.class.getName())
                .setMaster(master);
            JavaSparkContext sc = new JavaSparkContext(conf);
            sc.setLogLevel("ERROR");
           // System.setProperty("hadoop.home.dir", "c:\\winutils\\");

             JavaPairRDD<HashTagHashTagPair,Float> scores = null  ;

             String inputFilePath = "Users/*";
             JavaRDD<String> userInfo;

             userInfo = sc.textFile(inputFilePath);


             JavaRDD<Tuple2<Long,Integer>> userFollowersTemp = userInfo.map(x->{


                String[] s = (x.split("\\|"));

                return new Tuple2<Long,Integer>(Long.valueOf(s[0]),Integer.valueOf(s[1]));

             });


             JavaPairRDD<Long,Integer>                    userFollowers                    =
userFollowersTemp.mapToPair(new
PairFunction<Tuple2<Long,Integer>,Long,Integer>(){

             @Override

             public Tuple2<Long, Integer> call(Tuple2<Long, Integer> t) throws Exception {

                return new Tuple2<Long,Integer>(t._1,t._2);

             }


             });

//           userFollowers.foreach(x->{

//             TaskContext tc = TaskContext.get();

//             System.out.println("userFollowers "+tc.taskAttemptId()+"   "+x);

//                });

//

             //Get the hashtag hashtag pairs with their similarity score from the files
previously written

             JavaRDD<Tuple2<HashTagHashTagPair,Float>> hhf = sc.objectFile("scores");
```

```
        scores = hhf.mapToPair(new
PairFunction<Tuple2<HashTagHashTagPair,Float>,HashTagHashTagPair,Float>(){
         @Override
         public Tuple2<HashTagHashTagPair, Float>
call(Tuple2<HashTagHashTagPair, Float> t) throws Exception {
           return new Tuple2<HashTagHashTagPair,Float>(t._1,t._2);
         }
        });



        //Filter RDD with the hashtag in question
        scores = scores.filter(new
Function<Tuple2<HashTagHashTagPair,Float>,Boolean>(){
         @Override
         public Boolean call(Tuple2<HashTagHashTagPair, Float> t1) throws Exception
{
          if(t1._1.getHashTag1().equalsIgnoreCase(hash) ||
t1._1.getHashTag2().equalsIgnoreCase(hash)){
             return true;
          }else
             return false;
         }


    });


    //Revert scores in order to sort them
    JavaPairRDD<Float,HashTagHashTagPair> revertedScores =
scores.mapToPair(new
PairFunction<Tuple2<HashTagHashTagPair,Float>,Float,HashTagHashTagPair>(){
```

```
            @Override
            public Tuple2<Float, HashTagHashTagPair> call(Tuple2<HashTagHashTagPair, Float> t) throws Exception {

                return new Tuple2<Float,HashTagHashTagPair>(t._2,t._1);
            }



      });



      revertedScores = revertedScores.sortByKey(false);
//      revertedScores.foreach(x->{
//              TaskContext tc = TaskContext.get();
//              System.out.println("reverted "+tc.taskAttemptId()+"   "+x);
//                  });

      //Take the top 10 similar hashtags
      List<Tuple2<Float,HashTagHashTagPair>> topTen = revertedScores.take(10);



      JavaRDD<Tuple2<Float,HashTagHashTagPair>> temp = sc.parallelize(topTen);


      //Make an rdd that has only the hashtags that we need eg the most similar ones
      JavaPairRDD<String,Float> finTopTen = temp.mapToPair(new PairFunction<Tuple2<Float,HashTagHashTagPair>,String,Float>(){
            @Override
            public Tuple2<String, Float> call(Tuple2<Float, HashTagHashTagPair> t) throws Exception {


                return new Tuple2<String,Float>(t._2.getHashTag1().equalsIgnoreCase(hash)?t._2.getHashTag2(): t._2.getHashTag1(),t._1);


            }


        });
//      finTopTen.foreach(x->{
//              TaskContext tc = TaskContext.get();
//              System.out.println("finTopTen "+tc.taskAttemptId()+"   "+x);
//                 });



        //Collect them as a java map in order to user them below.
        Map<String,Float> msf = finTopTen.collectAsMap();
//        msf.put(hash,new Float(1.2));
        HashMap<String,Float> finMsftemp1 = new HashMap(msf);
        finMsftemp1.put(hash,new Float(1.2));



        JavaRDD<Tuple2<String,Integer>> hallLikes = sc.objectFile("HashTagAllLikes");
        //edw 8a ftiaxtei ta maps me ta 11 hashtags pou psaxnoume kai ola tous ta likes kai shares.
        JavaPairRDD<String,Integer> topTenHashLikes = hallLikes.mapToPair(new PairFunction<Tuple2<String,Integer>,String,Integer>(){
            @Override
            public Tuple2<String, Integer> call(Tuple2<String, Integer> t) {
                return new Tuple2<String,Integer>(t._1,t._2);
            }

        }).filter(new Function<Tuple2<String,Integer>,Boolean>(){
            @Override
            public Boolean call(Tuple2<String, Integer> t1) throws Exception {
                if(finMsftemp1.get(t1._1) != null)
                    return true;

                return false;
            }

        });

        Map<String,Integer>        finTopTenLikesHashTagstemp        =
topTenHashLikes.collectAsMap();
        HashMap<String,Integer>     finTopTenLikesHashTags     =     new
HashMap(finTopTenLikesHashTagstemp);

        JavaRDD<Tuple2<String,Integer>>          hashShares          =
sc.objectFile("allHashTagsShares");
        JavaPairRDD<String,Integer>  topTenHashShares  =  hashShares.mapToPair(new
PairFunction<Tuple2<String,Integer>,String,Integer>(){
            @Override
            public Tuple2<String, Integer> call(Tuple2<String, Integer> t)  {
                return new Tuple2<String,Integer>(t._1,t._2);
            }


        }).filter(new Function<Tuple2<String,Integer>,Boolean>(){
            @Override
            public Boolean call(Tuple2<String, Integer> t1)  {
              if(finMsftemp1.get(t1._1) != null)
                  return true;


              return false;
            }


        });


        HashMap<String,Integer> hashAllShares = new
HashMap(topTenHashShares.collectAsMap());




        //we add as the 11th element the hashtag in question
          Map<String, Float> finMsf = sortByComparator(finMsftemp1, false);
          float sum = 0;
          int j = 2;
          for(int i = 1; i<(finMsf.size()); i++ ){
              sum = sum + (float)1/(float)j;
              j = j * 2;
          }
          sum = (float)1/(float)(1+sum);
          Map<String,Float> weightedHash = new HashMap<String,Float>();


        for(Map.Entry e:finMsf.entrySet()){
           weightedHash.put((String) e.getKey(), sum);
           sum = (float)sum/(float)2;
        }
        float sumtemp = 0;
        for(Map.Entry e: weightedHash.entrySet()){
           System.out.printf("%s %.15f \n",e.getKey(),e.getValue());
//           System.out.println(e.getKey()+" "+e.getValue());
           sumtemp += (float)e.getValue();
        }


        JavaRDD<Tuple2<HashTagUser,Integer>>              tempHashTagUserL              =
sc.objectFile("HashTagUserPairLikes");
        JavaPairRDD<HashTagUser,Integer> hashTagUserL;


             hashTagUserL                   =                   tempHashTagUserL.mapToPair(new
PairFunction<Tuple2<HashTagUser,Integer>,HashTagUser,Integer>(){
            @Override
            public  Tuple2<HashTagUser,  Integer>  call(Tuple2<HashTagUser,  Integer>  t)
throws Exception {
                return new Tuple2<HashTagUser,Integer>(t._1,t._2);
            }
             });
```

```
    JavaRDD<Tuple2<HashTagUser,Integer>>        tempHashTagUserS        =
sc.objectFile("HashTagUserPairShares");
    JavaPairRDD<HashTagUser,Integer> hashTagUserS;


      hashTagUserS         =           tempHashTagUserS.mapToPair(new
PairFunction<Tuple2<HashTagUser,Integer>,HashTagUser,Integer>(){
      @Override
      public Tuple2<HashTagUser, Integer> call(Tuple2<HashTagUser, Integer> t)
throws Exception {
        return new Tuple2<HashTagUser,Integer>(t._1,t._2);
      }
     });


    hashTagUserS                =                hashTagUserS.filter(new
Function<Tuple2<HashTagUser,Integer>,Boolean>(){
      @Override
      public Boolean call(Tuple2<HashTagUser, Integer> t1) throws Exception {


          return  (msf.containsKey(t1._1.getHashTag())) ? true:false;


      }


    });
     hashTagUserL               =                hashTagUserL.filter(new
Function<Tuple2<HashTagUser,Integer>,Boolean>(){
      @Override
      public Boolean call(Tuple2<HashTagUser, Integer> t1) throws Exception {
```

```
                return (msf.containsKey(t1._1.getHashTag()))?true:false;


            }


        });


//       hashTagUserL.foreach(x->{
//              TaskContext tc = TaskContext.get();
//              System.out.println("hashTagUserL "+tc.taskAttemptId()+"   "+x);
//                 });
//       hashTagUserS.foreach(x->{
//              TaskContext tc = TaskContext.get();
//              System.out.println("hashTagUserS "+tc.taskAttemptId()+"   "+x);
//                 });


        JavaPairRDD<HashTagUser,Tuple2<Integer,Integer>> likesSharesRddtemp;


        likesSharesRddtemp = hashTagUserS.join(hashTagUserL);




        JavaPairRDD<Long,HashTagLikesShares> likesSharesRdd;
        likesSharesRdd = likesSharesRddtemp.mapToPair(new
PairFunction<Tuple2<HashTagUser,Tuple2<Integer,Integer>>,Long,HashTagLikesShar
es>(){
            @Override
            public Tuple2<Long,HashTagLikesShares> call(Tuple2<HashTagUser,
Tuple2<Integer, Integer>> t) throws Exception {
```

```
                return new Tuple2<Long,HashTagLikesShares>(t._1.getUserId(),new
HashTagLikesShares(t._1.getHashTag(),t._2._1,t._2._2));
            }



        });
//      likesSharesRdd.foreach(x->{
//              TaskContext tc = TaskContext.get();
//              System.out.println("likesSharesRdd "+tc.taskAttemptId()+"   "+x);
//                  });
        JavaPairRDD<Long,Iterable<HashTagLikesShares>> tempHash =
likesSharesRdd.groupByKey();

//     tempHash.foreach(x->{
//              TaskContext tc = TaskContext.get();
//              System.out.println("tempHash "+tc.taskAttemptId()+"   "+x);
//                  });

        JavaPairRDD<Long,Tuple2<Iterable<HashTagLikesShares>,Integer>>
userHashSLF = tempHash.join(userFollowers);
//      userHashSLF.foreach(x->{
//              TaskContext tc = TaskContext.get();
        //              System.out.println("userHashSLF "+tc.taskAttemptId()+"   "+x);
                                          //                  });

        JavaPairRDD<Long,Float> influencers = userHashSLF.mapToPair(new
PairFunction<Tuple2<Long,Tuple2<Iterable<HashTagLikesShares>,Integer>>,Long,Flo
at>(){
```

```
            float score = 0;
            @Override
            public Tuple2<Long, Float> call(Tuple2<Long, Tuple2<Iterable<HashTagLikesShares>, Integer>> t) throws Exception {

               score = 0;
               t._2._1.forEach(x->{
                   int allLikes = finTopTenLikesHashTags.get(x.getHashTag());
                   int allShares = hashAllShares.get(x.getHashTag());
//                         score += (float) weightedHash.get(x.getHashTag()) * ( (weightA* ((float)x.getLikes()/(float) t._2._2 )) + ((weightB*( (float)x.getShares() / (float)t._2._2 ))) + weightC*t._2._2);
                   score += (float) weightedHash.get(x.getHashTag()) * ( (weightA* ((float)x.getLikes()/allLikes  ))
                                 + ((weightB*( (float)x.getShares() / allShares ))) + weightC*t._2._2);

               });
               return new Tuple2<Long,Float>(t._1,score);

            }

        }).filter(new Function<Tuple2<Long,Float>,Boolean>(){
            @Override
            public Boolean call(Tuple2<Long, Float> t1) throws Exception {
               if (Float.isNaN(t1._2)) return false;
               else return true;
            }
```

```
    });


     JavaPairRDD<Float,Long> inf = influencers.mapToPair((Tuple2<Long, Float> t) ->
new Tuple2<Float,Long>(t._2,t._1));
      inf.foreach(x->{
           TaskContext tc = TaskContext.get();
           System.out.println("inf "+tc.taskAttemptId()+"   "+x);
              });


     List<Tuple2<Float,Long>> m =  inf.sortByKey(false).take(10);
     JavaRDD<Tuple2<Float,Long>> mtemp = sc.parallelize(m);
     JavaPairRDD<Long,Float>         mFin         =         mtemp.mapToPair(new
PairFunction<Tuple2<Float,Long>,Long,Float>(){
         @Override
         public Tuple2<Long, Float> call(Tuple2<Float, Long> t) throws Exception {
              return new Tuple2<Long,Float>(t._2,t._1);
         }


     });
     mFin.foreach(x->{
           TaskContext tc = TaskContext.get();
           System.out.println("mFin "+tc.taskAttemptId()+"   "+x);
              });


  }
```

Code to identify most influential urls (example #bigdata)

```
  String master = "local[4]";


            /*
             * Initializes a Spark context.
             */
            SparkConf conf = new SparkConf()
                .setAppName(Example.class.getName())
                .setMaster(master);
            JavaSparkContext sc = new JavaSparkContext(conf);
           sc.setLogLevel("ERROR");
         System.setProperty("hadoop.home.dir", "C:\\winutils\\");


            JavaPairRDD<HashTagHashTagPair,Float> scores = null  ;


            JavaRDD<Tuple2<HashTagHashTagPair,Float>> hhf = sc.objectFile("scores");


            scores = hhf.mapToPair(new
PairFunction<Tuple2<HashTagHashTagPair,Float>,HashTagHashTagPair,Float>(){
             @Override
             public Tuple2<HashTagHashTagPair, Float>
call(Tuple2<HashTagHashTagPair, Float> t) throws Exception {
                return new Tuple2<HashTagHashTagPair,Float>(t._1,t._2);
             }
            });


            JavaRDD<Tuple2<HashTagUrl,Integer>> hastUrlLikes;
            hastUrlLikes = sc.objectFile("HastagUrlLikes");
```

```
        JavaPairRDD<HashTagUrl, Integer> hul;
        hul = hastUrlLikes.mapToPair(new
PairFunction<Tuple2<HashTagUrl,Integer>,HashTagUrl,Integer>(){
          @Override
         public Tuple2<HashTagUrl, Integer> call(Tuple2<HashTagUrl, Integer> t)
throws Exception {
           return new Tuple2<HashTagUrl,Integer>(t._1,t._2);
         }

        });

        JavaRDD<Tuple2<HashTagUrl,Integer>> hashtUrlNo;
        hashtUrlNo = sc.objectFile("HashTagUrlPair");

        JavaPairRDD<HashTagUrl,Integer> hut ;
        hut = hashtUrlNo.mapToPair(new
PairFunction<Tuple2<HashTagUrl,Integer>,HashTagUrl,Integer>(){
         @Override
         public Tuple2<HashTagUrl, Integer> call(Tuple2<HashTagUrl, Integer> t)
throws Exception {
           return new Tuple2<HashTagUrl,Integer>(t._1,t._2);
         }

        });
        scores = scores.filter(new
Function<Tuple2<HashTagHashTagPair,Float>,Boolean>(){
          @Override
          public Boolean call(Tuple2<HashTagHashTagPair, Float> t1) throws Exception
{
           if(t1._1.getHashTag1().equalsIgnoreCase("#bigdata") ||
t1._1.getHashTag2().equalsIgnoreCase("#bigdata")){
              return true;
           }else
              return false;
          }

        });



    JavaPairRDD<Float,HashTagHashTagPair> revertedScores =
scores.mapToPair(new
PairFunction<Tuple2<HashTagHashTagPair,Float>,Float,HashTagHashTagPair>(){

          @Override
          public Tuple2<Float, HashTagHashTagPair>
call(Tuple2<HashTagHashTagPair, Float> t) throws Exception {

              return new Tuple2<Float,HashTagHashTagPair>(t._2,t._1);
          }



    });
    revertedScores = revertedScores.sortByKey(false);
    revertedScores.foreach(x->{
            TaskContext tc = TaskContext.get();
            System.out.println("reverted "+tc.taskAttemptId()+"   "+x);
               });

    List<Tuple2<Float,HashTagHashTagPair>> topTen = revertedScores.take(10);

    JavaRDD<Tuple2<Float,HashTagHashTagPair>> temp = sc.parallelize(topTen);

    JavaPairRDD<String,Float> finTopTen = temp.mapToPair(new
PairFunction<Tuple2<Float,HashTagHashTagPair>,String,Float>(){
          @Override
          public Tuple2<String, Float> call(Tuple2<Float, HashTagHashTagPair> t)
throws Exception {

             return new
Tuple2<String,Float>(t._2.getHashTag1().equalsIgnoreCase("#bigdata")?t._2.getHashT
ag2():t._2.getHashTag1(),t._1);

          }

    });
    finTopTen.foreach(x->{
            TaskContext tc = TaskContext.get();
            System.out.println("finTopTen "+tc.taskAttemptId()+"   "+x);
               });
```

```
Map<String,Float> msf = finTopTen.collectAsMap();


hut = hut.filter(new Function<Tuple2<HashTagUrl,Integer>,Boolean>(){
      @Override
      public Boolean call(Tuple2<HashTagUrl, Integer> t1) throws Exception {


            Float temp = null;
            temp = msf.get(t1._1.getHashTag());
            return temp != null ?true:false;
      }


});


hul = hul.filter(new Function<Tuple2<HashTagUrl,Integer>,Boolean>(){
      @Override
      public Boolean call(Tuple2<HashTagUrl, Integer> t1) throws Exception {


         Float temp = null;
         temp = msf.get(t1._1.getHashTag());
          return temp != null ?true:false;




      }


});
```

```
    hul.foreach(x->{
        TaskContext tc = TaskContext.get();
        System.out.println("hup: "+tc.taskAttemptId()+"   "+x);
          });

    JavaPairRDD<String,Integer> hulTrans = hul.mapToPair(new
PairFunction<Tuple2<HashTagUrl,Integer>,String,Integer>(){
       @Override
       public Tuple2<String, Integer> call(Tuple2<HashTagUrl, Integer> t) throws
Exception {

          return new Tuple2<String,Integer>(t._1.getUrl(),t._2);
       }

    });
    JavaPairRDD<String,Integer> hutTrans = hut.mapToPair(new
PairFunction<Tuple2<HashTagUrl,Integer>,String,Integer>(){
       @Override
       public Tuple2<String, Integer> call(Tuple2<HashTagUrl, Integer> t) throws
Exception {

          return new Tuple2<String,Integer>(t._1.getUrl(),t._2);
       }

    });
```

```
        hutTrans = hutTrans.reduceByKey((occ1, occ2) -> occ1 + occ2);
        hulTrans = hulTrans.reduceByKey((occ1, occ2) -> occ1 + occ2);
        hutTrans.foreach(x->{
                TaskContext tc = TaskContext.get();
                System.out.println("hutTrans: "+tc.taskAttemptId()+"   "+x);
                    });
        hulTrans.foreach(x->{
                TaskContext tc = TaskContext.get();
                System.out.println("hulTrans: "+tc.taskAttemptId()+"   "+x);
                    });

          JavaPairRDD<String,Tuple2<Integer,Integer>> htuI = hutTrans.join(hulTrans);
            hut.foreach(x->{
                TaskContext tc = TaskContext.get();
                System.out.println("hun: "+tc.taskAttemptId()+"   "+x);
                    });
        htuI.foreach(x->{
                TaskContext tc = TaskContext.get();
                System.out.println("htuI: "+tc.taskAttemptId()+"   "+x);
                    });

        JavaPairRDD<String,Float> scoresFin = htuI.mapToPair(FINAL_SCORE);
        JavaPairRDD<Float,String>     scoresFinTemp     =     scoresFin.mapToPair(new
PairFunction<Tuple2<String,Float>,Float,String>(){
              @Override
              public Tuple2<Float, String> call(Tuple2<String, Float> t) throws Exception {
                    return new Tuple2<Float,String>(t._2,t._1);
              }


        });
         List<Tuple2<Float,String>> m =  scoresFinTemp.sortByKey(false).take(10);


        JavaRDD<Tuple2<Float,String>> mtemp = sc.parallelize(m);
         JavaPairRDD<String,Float>      mFin      =      mtemp.mapToPair(new
PairFunction<Tuple2<Float,String>,String,Float>(){
              @Override
              public Tuple2<String, Float> call(Tuple2<Float, String> t) throws Exception {
                    return new Tuple2<String,Float>(t._2,t._1);
              }


         });


         mFin.foreach(x->{
                TaskContext tc = TaskContext.get();
                System.out.println("mFin: "+tc.taskAttemptId()+"   "+x);
                    });
                                                 }
```

REFERENCES

[1] Agarwal, N., Liu, H., Tang, L., & Yu, P. S. Identifying the influential bloggers in a community. In: Proc. Intl Conf. on Web Search and Web Data Mining (WSDM '08), pp. 207-218, ACM (2008)
[2] Ahmed, S. & Ezeife, C. I. Discovering influential nodes from trust network. In: Proc. 28th Annual ACM Symposium on Applied Computing (SAC '13), pp. 121-128. ACM (2013)
[3] Anger, I. & Kittl, C. Measuring influence on Twitter. In: Proc. 11th Intl Conf. on Knowledge Management and Knowledge Technologies (i-KNOW '11), ACM (2011)
[4] Apache, https://spark.apache.org/, access 28/6/2018
[5] Bagchi, S. Performance and Quality Assessment of Similarity Measures in Collaborative Filtering Using Mahout. Procedia Computer Science, Vol. 50, 229 – 234 (2015)
[6] Bento, C. Finding influencers in social networks. Dissertation, Instituto Superior Technico, Universidate Tecnica de Lisboa (2012)
[7] Chen, W., Wang, Y., Yang, S. Efficient influence maximization in social networks. In: Proc. 15th ACM SIGKDD Intl Conf. on Knowledge discovery and data mining (KDD '09), pp. 199-208, ACM (2009)
[8] De Meo, P., Ferrara, E., Fiumara, G., Provetti, A. Improving recommendation quality by merging collaborative filtering and social relationships. In: Proc. 11th Intl Conf. on Intelligent Systems Design and Applications (ISDA 2011), pp. 587-592, IEEE (2011)
[9] Ekstrand, M. D., Riedl, J. T., Konstan, J. A. Collaborative Filtering Recommender Systems. Found. Trends Hum.-Comput. Interact. 4 (2) 81-173 (Feb. 2011)
[10] Epinions, http://www.epinions.com/, access 28/6/2018
[11] Goyal, A., Bonchi, F., Lakshmanan, L. Learning influence probabilities in social networks. In: Proc. 3rd ACM Intl Conf. on Web search and data mining (WSDM '10). pp. 241-250, ACM (2010)
[12] Guo, G., Zhang, J.,Thalmann, Basu, A., Yorke-Smith, N. From Ratings to Trust: an Empirical Study of Implicit Trust in Recommender Systems. In: Proc. 29th ACM Symposium on Applied Computing (SAC), pp. 248-253 (2014)
[13] InfluenceTracker, http://www.influencetracker.com/, access 28/6/2018
[14] Klout, https://klout.com/, https://www.lithium.com/products/klout, access 28/6/2018
[15] Koutrouli, E., Kanellopoulos, G., Tsalgatidou, A. Reputation Mechanisms in on-line Social Networks: The case of an Influence Estimation System in Twitter. In Proc. SouthEast European Design Automation, Comp. Engineering, Computer Networks and Social Media Conference, pp. 98-105, ACM (2016)
[16] Prawesh, S., Padmanabhan, B. The "Most Popular News" Recommender: Count Amplification and Manipulation Resistance. Info. Sys. Research 25 (3), 569-589 (Sep. 2014)
[17] Razis, G., Anagnostopoulos, I. InfluenceTracker: Rating the Impact of a Twitter Account. In: 10th IFIP Intl Conf. on Artificial Intelligence Applications and Innovations, pp. 184-195, Springer, Berlin (2014)
[18] Realtime Tweets, https://developer.twitter.com/en/docs/tweets/filter-realtime/overview, 28/6/2018
[19] Riquelme, F., González-Cantergiani, P. Measuring user influence on Twitter. Inf. Process. Manage. 52 (5) (Sep. 2016), 949-975
[20] Ruohomaa, S., Kutvonen, L. Koutrouli, E. Reputation Management Survey. In: Proc. 2nd Intl Conf. on Availability, Reliability and Security (ARES), pp. 103-111 (2007)
[21] Trusov, M., Bodapati, A.V, Bucklin, R. Determining Influential Users in Internet Social Networks. Journal of Marketing Research, 47 (4), 643-658 (Aug. 2010)
[22] Twitter Counter, https://twittercounter.com/pages/100, access 28/6/2018
[23] Twitter embedded timelines, https://dev.twitter.com/web/embedded-timelines, access 28/6/2018
[24] Twitter Rest API, https://dev.twitter.com/twitterkit/android/access-rest-api, access 28/6/2018
[25] Twitter, https://twitter.com, access 28/6/2018
[26] Varlamis I., Eirinaki M., Louta M. Application of Social Network Metrics to a Trust-Aware Collaborative Model for Generating Personalized User Recommendations. In: The Influence of Technology on Social Network Analysis and Mining, vol 6. Springer (2013)
[27] Yazdanfar, N., & Thomo, A. Link Recommender: Collaborative-Filtering for Recommending urls to Twitter Users, Procedia Computer Science, Vol.19, 412-419 (2013)
[28] Ye, S., Wu, S. F. Measuring message propagation and social influence on Twitter.com. In: Proc. 2nd Intl Conf. on Social Informatics (SocInfo'10), Springer, pp. 216-231 (2010)
[29] Jensen, Carlos & Davis, John & Farnham, Shelly. (2002). Finding others online: reputation systems for social online spaces.. 447-454. 10.1145/503376.503456.
[30] Ricci, Francesco & Rokach, Lior & Shapira, Bracha. (2010). Recommender Systems Handbook. 10.1007/978-0-387-85820-3_1.