INTERNATIONAL
HELLENIC
UNIVERSITY

Content-based Tweets Semantic Clustering and Propagation

Marios Aristotelis Michalakos

SID: 3301130014

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

NOVEMBER 2014
THESSALONIKI – GREECE

Content-based Tweets Semantic Clustering and Propagation

Marios Aristotelis Michalakos

SID: 3301130014

Supervisors:	Christos Berberidis
	Nick Bassiliades
Supervising Committee Members:	Assoc. Prof. Name Surname
	Assist. Prof. Name Surname

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

NOVEMBER 2014

THESSALONIKI – GREECE

Abstract

In this thesis our goal was to develop a methodology in order to cluster a set of tweets based on their semantic context. We have used probabilistic topic modeling techniques such as Latent Dirichlet allocation in order to extract topics from our dataset and then we applied several natural language methods in order to automatically generate semantically meaningful and grammatically correct phrases, as candidate labels for our extracted topics, aiming at creating an objective method for topic labeling. Developing a scoring function in order to assign the most semantically similar labels to our extracted topics was an essential part to our research that has helped us to assign the most relevant labels to each topic. Then we have generated the Twitter graph and used community detection algorithms in order to analyze each community topic of interest. This way we have been able to record the propagation of certain topics in our graph and we have been able analyze the topics of interest in each community in our graph. Using visualization layout algorithms was also essential in order to provide meaningful visualizations of our networks. We have created datasets that was populated using Twitter's API and we have used open source tools in order to develop the software implementation of this method and a fully working prototype has been developed. Our research can be used as a valuable asset for modern market analysis from companies.

Acknowledgements

I would like to thank my supervisor, Christos Berberidis who patiently guided me during the process of my thesis and helped me to accomplish my research goals by giving me valuable advices.

I am dedicating this thesis to my brother Christos for his genuine support and to my parents.

Contents

1. Introduction

With the explosion of social media in the last decades and the rapid growth of micro-blogging services, an inexhaustible stream of data is produced every day. Billions of individuals all over the world interact with each other and generate mountains of data on various subjects. More than 30 billion tweets by year are produced on Twitter, allowing us to explore this ocean of data in order to understand and predict rare sociological events, reveal hidden patterns, detect trends or events and analyze them, perform sentiment analysis, observe how communities are formed, classify and uncover latent topics of discussion in communities and serve the needs of society. Twitter and similar micro-blogging services has drawn the attention of researchers all over the world the last few years to study, experiment and develop new techniques under the purview of data mining. The need for new and suitable text mining algorithms has emerged as scientists from different disciplines are trying to extract knowledge from large-scale social media data. Using Data mining, we can use techniques that can analyze massive unprocessed sources of data and extract information. For example, for a car sales company, interesting extracted information from social media data would be how likely is for an individual to purchase a certain car brand based on his personal data.

In our project, Twitter search API is going to be used which will allow us to retrieve tweets in JSON format using multiple parameters that include certain keywords, hashtags, locations and time boundaries. One of the biggest challenges is the nature of Twitter data, which is noisy and unstructured. A lot of pre-processing and data cleansing for noise reduction needs to be performed which involves utilizing natural language processing techniques, stopword removal, missing values handling, duplicate data removal, non-alpharethmetic characters removal, skip posts from accounts that are not in English in most cases and many more. In order to understand better our text collections we will need to utilize generative probabilistic topic models for our text corpora. Using Latent Dirichlet Allocation we will be able to generate a set of topics modeled as a mixture of words probabilities providing a suitable representation of our text collection. Although the discovered topics are often meaningful, a major challenge by all topic models is to accurately interpret the meaning of each topic. It is very difficult for a single individual to understand a topic in a “bag of words” with a distribution over these words, especially if the related topic is specialized in certain field. This is why we will implement a method of representing these topics in a concise enough manner that captures the meaning of the topic and allows for a human to understand it. In order to be able to label subjectively these extracted topics we will need to develop a technique that can automatically generate semantically meaningful phrases as labels for our topics using natural language techniques such as chunking/shallow parsing. The development of a scoring mechanism for the phrases as topic labels need also to be implemented in order to choose the most semantically meaningful phrases for our topics with the best possible accuracy.

The fact that social networks are gaining momentum exponentially makes social network analysis a field of study that has a lot to offer. Despite that analyzing data arising from social network platforms such as Twitter is a computationally intensive affair we can adapt techniques from graph theory and embed them into our methodology, suited for our data. Representing our collected data using graphs as networks is essential for us in order to discover interesting patterns, or relations between certain nodes (users). Making sense of individuals in our extracted network will involve the representation of a user as a node and an interaction as an edge. In social theory it is widely approved that the friendship of an individual affects how influential that individual is. However we will have to make different assumptions and explore different territories in order to reveal useful information among individuals. In order to extract community structure in our large networks we will use community detection methods that are based on modularity optimization. The fact that the typical size of large networks such as social networks includes millions or billions nodes interacting with each other makes the community detection in these networks a challenging task. On the other hand, social networks have a lot of natural structural properties, which can be leveraged for designing more effective algorithms. After setting the scope of social network analysis, we establish some general principles for social network visualization. Extracting the general topic of discussion for each community will help us to understand and identify the interests of each community. By assigning this label then in the most influential nodes in each community we will be able to visualize the topics of discussion in our network.

2. Literature Review

2.1 Introduction

Social networks have gained extraordinary attention in the last years and have become the universal communication mean that has thrived in making the world a global village. Accessing social network data such as tweets through Web 2.0 technologies has become easier and more accessible to researchers and individuals. Researchers had started to realize the importance of social network data as industries and companies had started to rely on social networks for knowledge extraction in order to discover the opinion of individuals or communities, trends, topics of interest and how and why communities are formed and under what circumstances. The interpretation of social network data is requiring us to develop methods for handling massive data, which are considerably noisy and dynamic. These issues can make social networks analysis a hard task. Researchers from the fields of data mining and machine learning have been developing techniques and methods in order to overcome these challenging tasks.

2.2 Sentiment analysis

A lot of methodologies and computer systems that can interpret Twitter data has already been implemented by researchers successfully that can help new researchers significantly to contribute on this ongoing field. A very interesting aspect of social media mining is sentiment analysis. Sentiment analysis is utilizing Natural Language Processing and Text Mining techniques to determine the attitude of a speaker or to classify the polarity of a given text in a document. Sentiment Analysis can be useful in many ways. It can be helpful for businesses and companies to be able to evaluate which of their products are popular and detect which of them are disliked from their costumers. This way companies can keep track of the sentiment changes of its costumers over time and take the appropriate actions in order to maximize their costumer satisfaction or needs and therefore maximize their profit as well. Having said that, sentiment analysis is not only used by companies for profit. In fact it has also been used to fight sociological problems such as suicide with the use of gender classification of Twitter users since Twitter does not obtain gender information from them. Due to the lack of gender information, most researchers do not consider the difference between men and women when performing sentiment analysis but it is known that risk factors for suicidal thoughts vary with gender and age. Machine learning techniques has been used in the past by some researchers to perform gender classification using Support Vector Machines algorithm which managed to achieve 70% of accuracy. Hyun Woo Kim (2010) in his thesis, analyses the use of four supervised learning algorithms as gender classifiers. Support Vector Machine, Naïve Bayes, Bayesian Logic Regression and Random Forest with the best accuracy offered by Random Forest with InfoGain classification algorithm. Under his experiments Random Forest managed to classify correctly 94% of the instances. Subsequently, after gender recognition is performed, Hyun Woo Kim is presenting ideas and concepts on how to create a suicide prevention system, capable of tracking suicidal thoughts and words from personal micro-blogs. This could be achieved by building a statistical suicide model that defines two sets of words, positive and negative. Using this modified sentiment analysis method on users by assigning relevant words of suicide to negative sentiment with weights between 0 and 1 and vice versa we can finally measure how likely is a person to approach the serious phenomenon and most probably to successfully prevent suicide.

2.3 Automatic Topic Labeling

Despite that natural language processing is used frequently to perform sentiment analysis it can also be used for other reasons. In our thesis we are going to use natural language processing techniques such as chunking or parsing in order to identify short phrases (chunks) in tweets. Using Natural Language Tool Kit Python's library, we are going to analyze every tweet in speech tags and then use these tags to make decisions of chunking according to a grammar that suites our needs. The reason behind the use of this technique is to generate phrases with grammatically correct meaningful phrases, in order to automatically label extracted latent topics from a Twitter dataset using Latent Dirichlet Allocation algorithm. Qiaozhu Mei, Xuehua Shen and Chengxiang Zhai (2007) are exploring a similar idea in their paper *"Automatic Labeling of Multinomial Topic Models"* by exploring the field of probabilistic topic modeling using multinomial distribution over words in text collections. Their differentiation between previous works in the field that has been conducted in the past has been addressed by the use of probabilistic approaches that can automatically label multinomial topic models in an objective way and not in a subjective way generated manually by humans. A major challenge in their approach was to accurately interpret the meaning of each topic and generate labels that are understandable. The authors' proposition towards this hurdle is the use of a probabilistic approach that automatically can label topics with meaningful phrases, since phrases are coherent and concise enough for users to understand, as opposed to sentences or single terms. By measuring then the "semantic distance" between a phrase and an extracted topic, the candidate labels from the extracted phrases can be assigned to the topics. The phrase generation as we explained can be approached using Chunking or Parsing but also with Ngram Testing, which it does not require training data but sometimes it does not produce linguistically meaningful phrases as Chunking/Parsing. In order to generate understandable semantic labels for each extracted latent topic from a given set, a semantic relevance scoring function is going to be utilized to rank labels by their semantic similarity to a topic model. This way the labels that are generated are understandable, semantically relevant, and discriminative across topics and of high coverage inside topics.

2.4 Trend detection

Societies, companies or individuals, always were interested in predicting the future in order to organize a strategy to benefit from it. A big interest in event and trend detection has come to light, as Twitter has become a rich source of information for detecting, monitoring and analyzing new stories and special events. Scientist in a worldwide scale has already conducted research within this field and numerous computer systems were implemented that can successfully perform trend detection. TwitterMonitor is a computer system developed by M. Mathioudakis and N. Koudas (2010) that can be used as an exploration tool for streaming information, analyzing or detecting emerging topics and trends in real time. A system like this could be very significant to news reporters, marketing professionals and opinion tracking companies in order to detect trend points that capture public's attention. As the authors are describing the methodology behind the system in their paper, TwitterMonitor identifies keywords that appear in high rate and cluster these keywords into trends based on co-occurrences. After a trend is identified, TwitterMonitor attempts to compose a more accurate description of the topic by employing context extraction algorithms such as PCA (principal component analysis) and SVD (singular value decomposition) and also by taking account of geographical origins of tweets. Finally, a chart is produced for each trend that depicts the evolution of its popularity over time and gets updated as long as the trend remains popular. They are also analyzing the architecture of TwitterMonitor, describing that the system consists of a back end which is connected with the Twitter API and receives a sample of the Twitter Stream with the 1/5 of the total tweets that are generated worldwide per day, and a front end that uses a webpage as a user interface that reports recent trends in real time. In our project, a fully working prototype is going to be developed were the topic extraction is going to be performed using LDA (Latent Dirichlet Allocation) algorithm which is a generative probabilistic model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. This way we will be able to automatically discover topics in a text collection. In our case our observations are words in a tweet and each tweet can be represented as a mixture of words. Based on these extracted topics, a set of tweets are going to be clustered based on their semantic context and then the goal will be to record the propagation of trending topics through the Twitter graph. Clustering of tweets using topic models can help to categorize them based on their properties. Analyzing then which of the topics are more trending and the way that they spread in the Twitter graphs can give us hints and reveal hidden information about upcoming events.

2.5 Knowledge Based Topic Labeling

A lot of researchers have proposed ways in order to enhance topic labeling using domain specific knowledge. In most cases, humans are not able to categorize or label topics that are related to a specific domain. For example for a set of tweets that are related to molecular biology only an expert in the specific domain would be capable of detecting the different topics among those tweets. With the rise of Wikipedia there is a corpus of data available that can aid us to better classify tweets or articles into a topic or a mixture of topics. In their paper, Ivan

Marcin and Sam Shiu (2012) are proposing the use of Wikipedia as a collection of human knowledge in order to interpret the content of articles and text from human conversation. With the use of massive Wikipedia and Twitter datasets the authors aim towards discovering useful information and relationships in them by classifying a test or articles in the dataset into a mixture of topics, associating them with the words that are distributed in Wikipedia topics. Their theory suggests that the events and the topics that are connected with these events are not isolated. These connections between topics change over time, which makes necessary the need for a machine generated way to identify these connections and link topics together. The authors propose a method to mine topics by extracting topics from Wikipedia with human assistance and then perform semantic analysis over both topics content and the Twitter data to generate a graph based on the correlation between topics and their related conversations on Twitter and cluster topics graphs to group topics related by their usage. The accuracy of this method matches the trend lines generated from Twitter interests and Google trends, which indicates its significant performance. This method has been evaluated through Google's search volume trend data by searching in Google for topics and then keeping track of the top topics for a given day by plotting both the trends generated by training data and Google trend data over time.

2.6 Entity Based Topic Discovery

Even though there have been a lot of successful methodologies for topic discovery, researchers has explored many different ways in the field due to the obstacles they often come across such as the nature of micro blogging data or handling real time data. In their paper, M. Michelson and S. A. Macskassy (2010) are introducing an entity based topic discovery method for Twitter tweets. Their research is focusing on topic discovery of particular Twitter users. The main idea is to conduct automatic generation of "topic profiles" for each Twitter user by finding the entities for the tweets of a user and then determine a set of categories that covers these entities. Taking account of the noisy nature of the micro blogging data is also very important task and makes the entity detection a tough assignment. As this paper confirms, these challenges are going to be addressed using Wikipedia as an entity knowledge base and the overall approach as a computer system implementation is going to be called Twopics, which is going to be used as a category discovery mechanism at first and then as an entity analysis tool in a user set of tweets trying to match the set of categories that defines the user's topic profile. A query on Wikipedia then is going to be performed in order to return a set of entity candidates. The goal is to choose the Wikipedia entity from a set of entity candidates that maximizes the overlap between contexts to accurately "guess" the category of a tweet. The experimental work of Twopics shows great performance and accuracy especially as the authors proposed with the use of a supervised Support Vector Machines (SVM) approach, despite the difficulty of use because of the training data as a supervised learning method.

In our project we are going to experiment with named entity recognition in text sentences using the corpus of Natural Language Tool Kit library in Python. The goal will be to track important words such as brand names, organizations, geo-locations and important persons in order to track important words in a tweet and build our *chunker* around these words to generate semantically meaningful and representative phrases for each tweet. As we explained earlier, these phrases then are going to be used as candidate labels for our extracted topics

2.7 Social Network Analysis

Graph theory has been widely used by researchers for social network analysis even on the early days of social network concepts. The approach of previous researchers in the field of social network analysis was to determine important nodes (users) and edges (interactions) in the network, for example how influential is a certain user. Influencers on social networks are considered the nodes (users) that have impact on the opinion of other nodes (users) and on their decision making on the network. Researchers had tackled these problems by implementing graph theory techniques and reducing the problem of large-scale datasets (such as social network data) by using data matrices as data representation of networks. The author Burt R. S. (2005) has used centrality measure as a mean to calculate the influence that forms clusters on social networks.

Moreover, Ghosh R and Lerman (2011) have used parameterized centrality metric approach to study the structure of social networks and to rank the connectivity of nodes. Their work has helped for the extension of α-centrality approach, which measures the number of alleviated paths that exist among nodes. . In our research we are going to develop a different method for measuring influence, which involves the ad-hoc community networks that are involved in topics of discussions that are not based on friendships or followers of a node (user).

2.8 Community Detection in Social Networks

It is really essential for social network analysis to develop techniques in order enable community discovery in large networks such as social networks. A community is generally considered to be a sub network in a larger network.

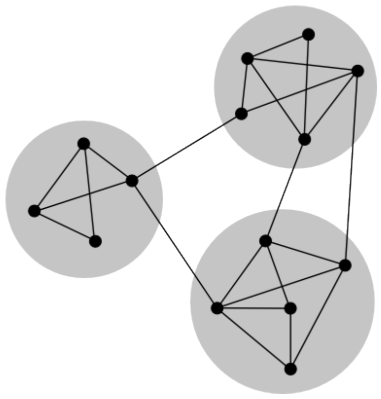


Figure 0: Social Network Community Structure example

The formation of communities under the scope of discussion over topics of interests is considered to be very important for knowledge discovery in social networks. Nodes (users) with similar interests will form communities with certain characteristics and patterns will start to emerge. The nature of communities in social networks is very complex and difficult to study and understand. The need for developing the appropriate tools in order to detect the behavior of network communities is crucial in order to study them and to extract useful information from them. In his research on social networks, M. Newmann (2010) has developed several clustering techniques to detect communities on social networks using hierarchical clustering. Using this technique he was able to cluster nodes in groups in a network and measured the strength of certain groups, which was used later to distribute the network into communities. Vertex clustering belongs to hierarchical clustering methods; graph vertices can be resolved by adding it in a vector space so that pairwise length between vertices can be measured.[22]

Structural equivalence measures of hierarchical clustering use a number of common network connections that is shared by two nodes. Two nodes on a social network with common friends (or followers) are more likely to be closer than two other nodes with less common friends (or followers).

It has been shown by Vincent D. Blondel, Jean-Loup, Renaud Lambiotte and Etienne Lefebvre (2010) in their work that the extraction of community structure of large networks is possible by using the proper methodology. Using their heuristic method, which is based on modularity optimization a large topology of interconnected nodes, can be analyzed into different communities. The problem of community detection requires partition of the network into communities of densely connected nodes. Modularity has been used to compare the quality of the partitions obtained. The proposed algorithm for modularity optimization can allow us to study networks of unprecedented size, which is very important for the dynamic nature of social networks. The proposed algorithm unfolds a complete hierarchical community structure for the network and each level of the hierarchy being given by the intermediate partitions found at each pass. The quality of the communities detected in their experiments is very good as measured by modularity. In our research we are going to implement this algorithm in order to discover communities in our extracted social networks. This will help us because of the size of our network, which is large, but also for the quality of the communities that we can extract.

3. Problem Definition

Mining the Social Media is an emerging field, which implements interdisciplinary concepts and theories, fundamental principles and state of the art algorithms. In order to develop sound data mining techniques for Twitter network analysis and to cluster our dataset based on the content of our data we need to define the problems we are going to face and to plan ahead in order to produce quality results and reach our research goals. The fact that our research is mainly addressed as a data mining problem we are going to adopt the general problem definition of data mining tasks and fit it in our research needs.

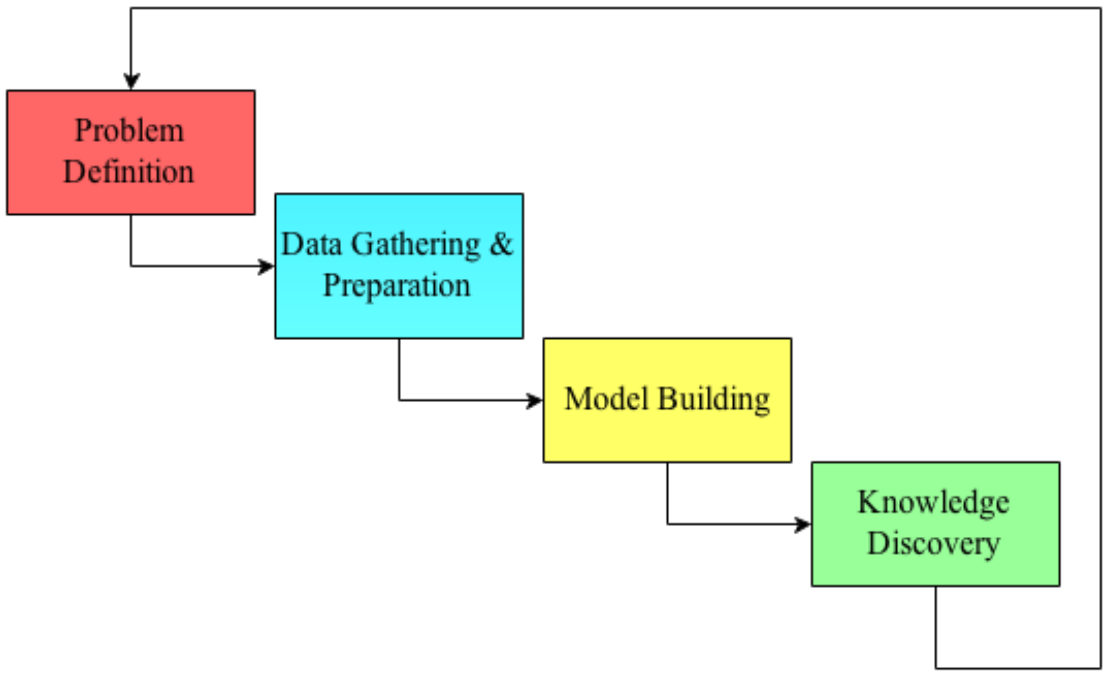


Figure 1: Problem Definition Diagram

3.1 Data-Gathering and Preparation

Data pre-processing is a crucial step in data mining and machine learning projects. A huge amount of irrelevant, noisy and unreliable data especially in micro-blogging platforms such as Twitter exists, which makes knowledge discovery a considerably challenging task and can easily lead to misleading results. This is why we need to address the main problems we are going to face which are going to require data preparation and a lot of filtering steps.

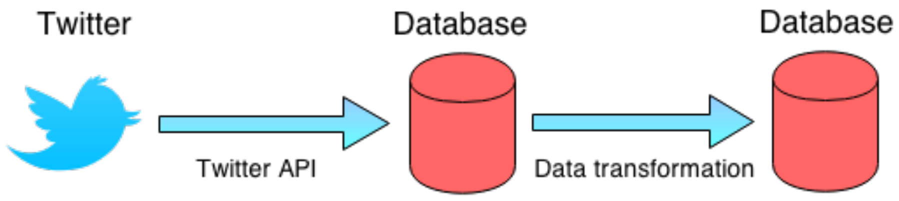


Figure 2: Data Gathering and Preparation

Obtain Necessary and Sufficient Data. The common method in order to obtain data from Twitter is to use application programming interfaces (APIs) specially designed for Twitter. Twitter allows a limited amount of data to be obtained daily from a developer account. This is why we need to make sure that our data is a reliable representation of the full available data in order to accomplish our research goals. We also need to consider the data format of tweets (JSON) and collect only necessary fields on a tweet that can only benefit our research without exhausting our dataset with useless information.

Data Management. We need to collect information using Twitter APIs and store them in a Database in order to proceed in data management methods and enable knowledge discovery in our database. Using our database management system we want to allow massive write performance, big amount of data storage, fast key-value data access, flexible schema and flexible datatype, document orientation, graphs, advanced data structures and ease of use. Twitter produces tweets that are in a JSON format. This is forcing us to use a NoSQL type of database system like MongoDB, which is an open-source document database system that can help us to interpret and control directly our streaming data without transforming it to suitable data types for an SQL like database management systems.

Noise Reduction. Noise reduction is an essential pre-processing step in the field of data mining and knowledge discovery in databases especially in unstructured data such as micro-blogging. Due to the nature of micro blogging data in social networking platforms such as Twitter, a precise noise reduction method need to be implemented in tweets in order to remove non valuable information without eliminating important data. Having said that noise reduction is a relative matter and a lot of experimentation needs to be done using pre processed corpus, stopwords removal, and natural language processing techniques to develop a satisfactory noise removal method for our datasets.

3.2 Topic Modeling

3.2.1 Topic Extraction

After the data collection and the data preparation we aim at discovering what topics these tweets are representing in our collection and how they are differentiate from each other. In order to do that we need to implement and develop topic modeling algorithms to help us understand, summarize and search these large electronic archives. Uncovering these latent topics in large collections is a challenging task, which involves probabilistic topic modeling techniques. Furthermore it can enable us to assign to each tweet the most relevant topic and organize them based on their topic. Various topic modeling approaches has been proposed by researchers such as Probabilistic Latent Semantic Analysis (PLSA), Hidden Topic Markov Models (HMM), Latent Dirichlet Allocation (LDA) and many more. In our thesis we are going to implement the most popular approach to topic modeling, which is Latent Dirichlet Allocation (LDA). This method treats the collections of documents (tweets in our case) as "bag of words" and assumes that the order of words can be ignored and that the text corpora can be represented by a co-occurrence matrix of words and documents. However this method hides pitfalls if we proceed to topic labeling manually and this is why we need to develop a wiser strategy for topic labeling. The fact that the user determines the number of topics raises the problem of finding an optimum number of topics for a given dataset. Unfortunately there is no "correct" number of topics to be extracted and there is not any sound solution to this problem except of using Hierarchical Dirichlet Processes (HDP-LDA) to model the Dirichlet admixture.

3.2.2 Topic Labeling

A major challenge in applying topic models in text collections is to label multinomial topics in an accurately way and to capture the essence of the topic in a few words. Delivering a more accurate and subjective way of automatic topic modeling could lead to richer and more profitable results. However finding a way to accurately label our extracted multinomial topics raises new problems and forces us to utilize natural language processing and understanding techniques in order to deal with them. What we want to achieve using these techniques is to produce a topic label l, for a topic model θ, that is a sequence of words, which is semantically meaningful and covers the latent meaning of θ. Under this definition a suitable type of a label l could be a sentence or a phrase. Based on the fact that our topic model is build upon tweet collections a more constistant and compaqt type of label such as phrases would be more suitable. Despite the fact that we may be able to extract phrases from our corpus we will also need to find a relieable way of assigning these labels to our extracted topics. The best way to achieve this would be by measuring the semantic similarity between the label and the topic model. For example given two labels l_1, l_2 that are both meaningful candidate labels, l_1 is a better label for a topic θ if $s(l_1,\theta) > s(l_2,\theta)$ where s is the relevance scoring function we need to develop in order to measure the semantic similarity between the label and the topic θ. With these definitions, the problem of topic model labeling can be defined as follows:

Given a topic model θ extracted from our dataset, the problem of single topic model labeling is (1) to identify a set of candidate labels $L = \{l_1,..,l_m\}$, and (2) to design a relevance scoring function $s(l_i,\theta)$. With L and s, we can then select a subset of n labels with the hisghest relevance scores $L_\theta = \{l_{\theta 1},...,l_{\theta n}\}$ for θ. [2]

3.3 Knowledge Discovery

To progress in our research after using various techniques to structure our data as we proposed previously we need to proceed to network measurements in order to represent, analyze and extract actionable patterns from our social media data. The fact that the world around us can be represented as a social network can reveal that trying to model such dynamic in nature systems can lead to many problems. Despite the complexity of this task by modeling such systems we can still gain useful information that can help us to understand how information is distributed in social networks and reveal hidden patterns regarding the information flow among users. Implementing mathematical structures such as graphs from graph theory is necessary in order to model relations in our data. We need to proceed into a second layer of processing our data using graphs and there are a lot of challenges in the nature of graphs that need to be addressed and overcome. As the data stored in our database grows and changes periodically we need to provide data summarization though visualization techniques, to identify important patterns and trends and act upon the findings. Insight derived from data mining can provide tremendous value to our goals and in the strategy we are going to follow.

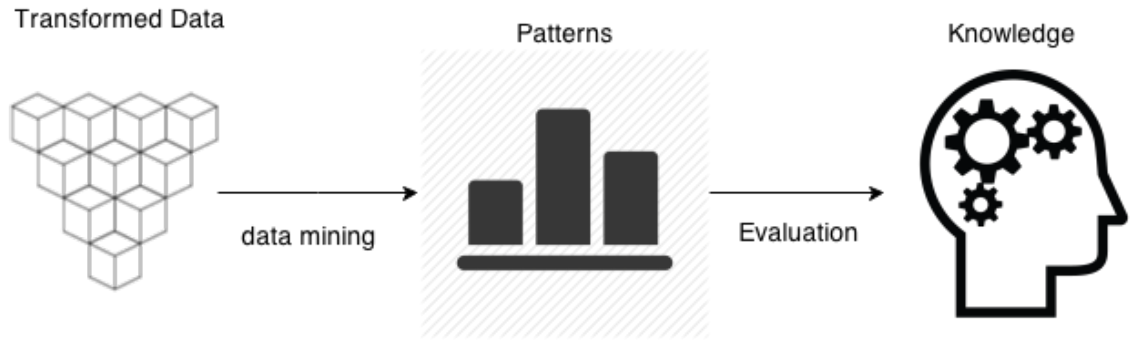


Figure 3: Knowledge Discovery

3.3.1 Network Analysis

Considering that Twitter can represent real life interactions and relationships as a platform, that can change dynamically through time, we need to make sure that in order to extract useful information from our network analysis we need to find a way to keep our dataset not exhausted from information that is not needed and that can mislead our network analysis. This is why we need to use our Twitter API wrapper wisely by filtering unnecessary data that can distort the uniformity of our network. Tracking abstract subjects that have a general meaning will lead to the disuse of the methods that we have developed. Depending on a certain amount of processing power, we need to track subjects and create samples that can represent valid real life interactions and allow patterns in our network to emerge. To generate a network from our data requires a definition for the interactions between the nodes (users). Creating edges between nodes based on the following of each node would not lead us anywhere. The connections (followers) of a node (user) is not required to follow the same interests of his followers or of the users he follows. This is why we need to examine the structure of a tweet format and find the best possible definition for an interaction in order to gain insights for certain topics of interests. We also need to answer questions like:

- Who are the most important people in a social network?
- Why do people interact with a certain amount of users in our network?
- How can we find interesting patterns in the content of our users?
- How can we identify communities in a social network?
- How we can measure the influence of individuals in a social network?

To answer questions like this we need to utilize graph theory techniques and represent users as nodes, interactions between them as edges and community formations as entities. Analyzing how communities are formed, how they evolve and how the qualities of detected communities are evaluated is also a crucial task. We also need to define who is an influential individual specifically for the needs of our network and how this influence can be reflected to other individuals.

3.3.2 Community Detection

We need to develop and implement a simple method in order to extract the community structure of our social network without loosing the quality of the communities that are detected. A lot of research in the field has been conducted and the problem has been leveraged to a modularity optimization problem and it is approached by Louvain method. Unfortunately modularity optimization is a problem that is computationally hard and so approximation algorithms are necessary when dealing with large networks. This is why we need to implement a fast approximation algorithm for optimizing modularity in a large network like the one in our dataset.

Furthermore the problem of community detection requires the partition of networks into communities of densely connected individuals (users), with the individuals belonging to different communities being only sparsely connected. Precise formulations of this optimization problem are known to be computationally intractable.[7] Several algorithms have therefore been proposed to find reasonable good partitions in a reasonably fast way. This search for fast algorithms has attracted much interest in recent years due to the increasing availability of large network data sets such as social networks and the impact of networks on everyday life.

Validation of the community would be necessary in our thesis in order reassure the effectiveness of the methods that we have developed. This could suggest as to create datasets that their community structure can be predetermined like for example political parties however we will still need to experiment due to the dynamic and complex nature of social network data.

3.3.3 Community Analysis

Community detection is not enough in order for us to extract semantic conclusions about our network. Identifying the community's topics of interest manually is not an option since it could be impossible for a human to accurately estimate them based on the community's nodes' data (tweets). In order for us to acquire useful information about these extracted communities that we have been able to detect, we will need to re-use the topic modeling methods that we have developed and proceed with the development of an automatic method for community labeling. Re-transforming and ordering our data based on the communities that have been detected could help us determine the thematic topics of interest in a distribution of words in our network.

Developing a scoring function to be applied in each of our extracted communities would be essential and challenging, however we could re-use the scoring function that we have developed for our topic labeling method with some modifications in order to be adjusted in the problem of community labeling. Assigning phrases as labels in the communities instead of topic's bag of words models could also provide a better representation. We also have to be able to minimize the amount of unimportant communities in our network. For example by keeping communities that are isolated from the others could distort our results. Finding a method for measuring the importance of a network is a challenging task what we would have to overcome by implementing graph theory techniques.

3.3.4 Visualization

Data visualization is a modern equivalent of visual communication and it is viewed as modern branch of descriptive statistics but also as a grounded theory development tool. It involves the creation and study of the visual representation of data, meaning information that has been abstracted in some schematics form, including attributes or variables for the units of information. The primary goal of data visualization in our thesis would be the efficient extraction of knowledge. Effective visualization can help us to analyze and make better sense of the processed data in our database. It can also make our data more accessible, understandable and usable by reducing their complexity. The fact that data visualization is not only science but also art, suggests us that there is not any objective way or method of producing efficient methods for data visualization and creates a lot of opportunities for visualization and algorithm design.

The obvious way to proceed in our research is to visualize our generated network using graph drawing, which will create a pictorial representation of the vertices and edges of a graph. The main problem by simply drawing our network is that the final graphic representation would be completely useless as the position of each node and community will not correspond to a semantically meaningful visualization. Applying different layout algorithms to our graph could lead to better output providing aesthetically improved visualizations. Layout of social networks is contingent on many factors. There are many different graph layout strategies available but it is widely suggested to proceed with force-based layout systems are most commonly used for social networks. This is likely because of their generality, simplicity, adaptability, and above all their availability. While force-directed methods generally perform well in separating clusters in graphs with varying local density, these methods are particularly troubled by small distances and skewed degree distributions. Forced based layout systems allow the graphs to be modified by continuously moving the vertices according to a system of forces based on physical metaphors related to systems of springs or molecular mechanics. Typically, these systems combine attractive forces between adjacent vertices with repulsive forces between all pairs of vertices, in order to seek a layout in which edge lengths are small while vertices are well separated. These systems may perform gradient descent based minimization of an energy function, or they may translate the forces directly into velocities or acceleration for the moving vertices. [17]

4. Methodology

4.1. Natural Language Processing and Understanding

Natural language processing (NLP) is a field of information science, artificial intelligence and linguistics that deals with the interaction between human languages and computers. Statistical natural language processing uses stochastic, probabilistic and statistical methods, especially to resolve difficulties that arise because longer sentences are highly ambiguous wen processed with realistic grammars, yielding thousands or millions of possible analyses. Methods for disambiguation often involve the use of large corpora and Markov models. Probabilistic model consists of a non-probabilistic model plus some numerical quantities improving significantly the developing of natural language processing systems. Challenges in natural language processing involve natural language understanding that can enable computers to derive meaning from human or natural language input. The natural language generation systems convert information from computer databases to human readable language. In artificial intelligence natural language understanding is a subtopic of natural language processing that deals with machine reading comprehension. The main goal of Natural language understanding systems is to convert samples of human language into formal forms as trees (parse trees) of first order logic that is easier for computer programs to handle. A computer must be able to model structure of words (syntax) in order to understand a sentence, and a model of syntax is necessary to produce grammatically correct sentences. [12] The process of disassembling and parsing input is more complex than the reverse process of assembling output in natural language generation because of the occurrence of unknown and unexpected features in the input and the need to determine the appropriate syntactic and semantic schemes to apply to it, factors which are pre-determined when outputting language.

In theory natural language processing is a very attractive method of human computer interaction. Old systems like SHRDLU [23] (developed by Terry Winograd), which operate in restricted “block worlds” with restricted vocabularies performed extremely well, leading researchers to excessive optimism, which was quickly lost when systems had to deal with the ambiguity and complexity of real world problems. In most languages one word can relate with many different things and this is why we must be able to choose what is the real meaning of a word, depending on the context that it appears. The grammar is usually not unambiguous with respect to e.g. parse trees that can be extracted from a sentence. The fittest word is going to be used based on the semantics of a collection of text. Various attempts at processing natural language from English corpus have been conducted through the years. Some attempts have not resulted in systems with deep understanding, but have improved the overall system usability. Despite that natural language processing faces a lot of challenges and the progress in these topics are getting slower such as text summarization (take input as a text document and try to condense them into a summary) or machine dialog system (understanding user inputs and respond accordingly), other topics are gaining momentum such as part of speech tagging, named entity recognition, sentiment analysis, conference resolution, word sense disambiguation, parsing and machine translation. Natural Language Toolkit (NLTK) for Python can provide us easy-to-use interfaces and a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing and many more.

4.1.1 Data Transformation

In order to allow our data to be processed through our *Part-Of-Speech Tagger (POST)* we need to make sure that our data does not contain any unnecessary information that can mislead our results. This kind of information is *stopwords*, HTTP links and non-English text. Utilizing natural language processing and understanding techniques in order can help us to overcome these problems.

Tokenization: Tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens. The list of tokens can become input for further processing such as parsing or text mining. To control the consistency of every word in a tweet we will utilize a tokenizer from Natural Language Tool Kit in Python’s library (NLTK) and transpose a tweet into an array of tokens. For example using a part from our tokenizer in our code in Python we get:

```
with open('tweets.csv','rU') as csvfile:
tweetreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
        for row in tweetreader:
                x=' '.join(row)
                tokens = nltk.word_tokenize(x)

                print (tokens)

                y=' '.join(tokens)
                tweetList.append(y)
```

For example having a tweet in our dataset like this example:

tweet='@user did you see the dog's face? #dogs'

After the tokenization procedure the result would result to an array like this:

tokens=[('@'),('user'),('did'),('you'),('see'),('the'),('dog'),('s'),('face),('?')('#")('dogs')]

It is easier now for our token array to be analyzed for stop word removal, HTTP cleansing, or English dictionary checking.

Stopwords removal and dictionary checking: Stopwords are words, which are filtered out of a dataset before or after the processing of natural language data (tweets). There is not one definite list of stop words which all tools use and such filter is not always used and this is why we need to determine the most common stop words for a tweet. Using a set of stop words that appear frequently in an English document with words such as: the, a, in, to, some, that, is, are etc. and also stop words especially for tweets such as: lmao, wow, bb, gd, lol etc. we can proceed to stopwords removal phrase. In the following code we read our stop words in a list variable and then removing all stop words detected in a tweet. We are also removing any word that appears more than once and then proceed to an English dictionary checking creating this way an ideal corpus for applying data mining or natural language processing techniques.

```
output = open('Stopwords_for_Tweets.txt', 'r')
stoplist=set(output.read().split())
texts = [[word for word in tweet.lower().split() if word not in stoplist]
for tweet in tweetList]
#remove words that appear only once
all_tokens = sum(texts, []) tokens_once = set(word for word in
set(all_tokens) if all_tokens.count(word) == 1)
```

```
texts = [[word for word in text if word not in tokens_once]
                                for text in texts]
dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]
```

Link removal: Removing links from tweets is also essential to clean our data from useless information that can distort our outcome. By assigning the string '`http`' to a variable named `urlone` we can use the following code, which can process every tweet as a string and remove any http link by checking if this variable exists in a string.

```
urlone='http:'
 with open(tweets.csv','rU') as csvfile:
        tweetreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
        for row in tweetreader:
        #Removing all links-URLs from Tweets
                for word in row:
                        if urlone in word:
                                row.remove(word)
```

4.1.2 Part-of-speech tagging (POST)

Part-of-speech tagging (POST) is the process of marking up a word in a text corpus as corresponding to a particular part of speech, based on both its definition, as well as its context. It involves the identification of words as nouns, verbs, adjectives, adverbs, etc. The fact that some words can represent more than one part of speech at different times makes it difficult for us to simply apply a part-of-speech algorithm in our corpus. [12] A sentence illustrating this problem would be: "*I can can a can*". Most part-of-speech taggers are trained using a treebank which is parsed text corpus that annotates the semantic structure of a sentence and in most cases uses newswire domain, such as the Wall Street Journal corpus of the Penn Treebank. Tagging performance degrades on out-of-domain data, and Twitter data poses additional challenges due to the conversational nature of the text, the lack of conventional orthography, and 140-character limit of each message ("tweet"). However using regular expressions and a tokenizer to trim our data can help us to improve the performance of the part-of-speech tagger algorithms dramatically. Classifying our text collection into word classes such as nouns, verbs, adjectives etc. is useful for the development of many natural language processing methods that we are going to tackle in this thesis such as chunking or shallow parsing. Using Natural Language toolkit (NLTK) we can use our already trimmed tweets as an input and proceed to a part-of-speech tagging of each word in every tweet and name these words accordingly depending on their word classes. By simply running this code below we can output the speech tag attached to each word like this for example:

```
for tweet in texts:
        tagged = nltk.pos_tag(tweet)
```

Input: tweet='use the select function on the sockets'

Output: ('use', 'VB'), ('the', 'DT')('Select', 'VB'), ('functions','NN'), ('on', 'IN'), ('the', 'DT'), ('sockets, 'NNS')

Transforming tweets using part-of-speech taggers can enable us to use our data in order to develop a *chunker* as an information extraction mechanism in order to extract the needed chunks from a tweet that can be used later as candidate labels for our topics.

4.1.3 Chunking/Shallow Parsing

As we explained in the problem definition section of our thesis, the extraction of phrases using our dataset is important in our research in order to develop an automatic way of creating labels for our extracted topics. To generate labels that are understandable, semantically relevant, discriminative across topics and of high coverage of each topic, we first extract a set of understandable candidate labels in a preprocessing step and then design a relevance scoring function to measure the semantic similarity between a label and a topic. Next we will be able to propose label selection methods to address the inter-topic discrimination and intra-topic coverage problems. Chunking (Shallow Parsing) is a common technique in NLP, which aims at identifying short phrases, or "chunks" in text. A chunker often operates on text with part-of-speech tags, and uses tags to make decisions to of chunking according to some grammar, or through learning from labeled training sets. We aim at extracting chunks as phrases that appear frequently in our dataset of tweets according to some grammar that we have to define. The advantage of using a NLP chunker is that the phrases that we can extract will be grammatical and meaningful. The accuracy of the chunker is highly affected by the domain of the text collection. [2] In our case, a corpus of tweets could result to a lot of "bad" chunks because of the dynamic nature of tweets and the lack of supervision in terms of orthography or syntactic completeness, however by cleaning our data from unwanted information our chunker will generate more meaningful phrases and overcome the noisy nature of micro-blogging data. In order to allow our chunker to track grammatical patterns in our dataset we will need to utilize the functionality of *Regular Expressions*. A *Regular Expression* (Regex) is a sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching. Each character in a regular expression is either understood to be a metacharacter with its special meaning, or a regular character with its literal meaning. They can be used to identify textual material of a given pattern. The pattern sequence itself is an expression that is a statement in a language designed specifically to represent prescribed targets in the most concise and flexible way to direct the automation of text processing of text files. The fact that we want to use the extracted chunk/phrase as a label for our topics suggests us that it is wise to use a grammar that produces small and concise types of chunks. Defining the grammar of our chunker using regular expressions can help us to experiment with the results using tree drawings from Python. The best grammar patterns that we have been able to define for our dataset are:

- `Phrase_Pattern_A: {<RB.?>*<VB.?>+<NNP>}` Using this grammar pattern we will be able to identify patterns in our dataset that have zero or more adverbs (and any kind of adverb like comparetive or superlative) followed by zero or more verbs (and any kind of verb like past tense, gerund, past participle, present, third person etc.) followed by exacctly one proper noun. Using this kind of grammar pattern we are trying to detect the what words are modifying a proper noun in a tweet.
- `Phrase_Pattern_B: {<JJ.?>*<NNP>}` Using this type of grammar pattern we can easily extract chunks that consists of zero or more adjectives (including all type of adjectives like comparetive or superlative) and one proper noun. Despite that this method looks simple can produce consice phrases.
- `Phrase_Pattern_C:`
  `{<NN\w?>*<DT\w?|NN\w?>*<JJ\w?|VG\w?>+<NN\w?>+<JJ\w?|VG\w?>*}`

Using a more ambitious pattern like this we are aiming at discovering more accurate phrases for a tweet, however statistically could lead to a worst grammar pattern overall. We are using one or more nouns (and any kind of noun) followed by one or more determiner (any type of determiner) or another proper noun (and any type of noun), followed by 1 or more adjective (and any type of adjective) or any type of verb followed by one or more nouns (and any type of nouns) followed by zero or more adjectives (any type of adjective) or any type of verb. Using Python's NLTK library we can create a function that could process our tweets using NLTK's RegexpParser, Tokenizer, pos_tagger and then draw a parse tree to represent our data.

```
def processLanguage():
try:
        for item in tweetlist:
        tokenized = nltk.word_tokenize(word)
        tagged = nltk.pos_tag(tokenized)
        cGram = r"""PhraseA: {<RB.?>*<VB.?>+<NNP>}"""
        #cGram = r"""PhraseB: {<JJ.?>*<NNP>}"""
        #cGram = r"""PhraseC:
{<NN\w?>*<DT\w?|NN\w?>*<JJ\w?|VG\w?>+<NN\w?>+<JJ\w?|VG\w?>*} """
        cParser = nltk.RegexpParser(cGram)
        chunked = chunkParser.parse(tagged)
        print chunked
        chunked.draw()
except Exception, e:
        print str(e)
        time.sleep(0.5)
```

Using the function `processLanguage()` and specifically with `PhraseC` and `PhraseB` type of grammar pattern in a small toy database that we have created tracking tweets about economy we get some of the following examples. We are going to analyze the results in more depth in the next chapter. For these tweets we have been able to extract the following chunks/phrases:

- Arab Monetary Fund ready to contribute to reform of Algerian financial system #Economy. The Arab Moneta... http:\/\/t.co\/2ehne4GzU5

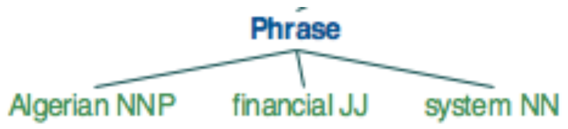


Figure 4: Phrase generation using PhraseC grammar

- @CoryBooker Women R used against the political economy of recession &amp; unemployment numbers; paying us less 4 our work help job nbrs improve

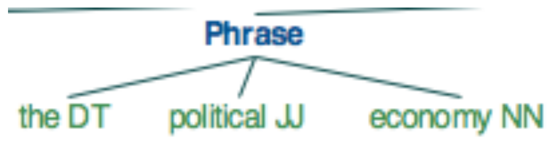


Figure 5: Phrase generation using PhraseC grammar

- Probably so RT @Train5829: Are you really middle class? http:\/\/t.co\/GPGBXHksKJ via @CNNMoney

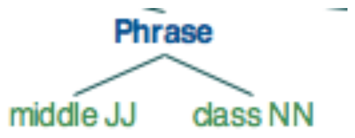


Figure 6: Phrase generation using PhraseB grammar

Generating these phrases from our dataset will become useful when we extract topics. By developing a scoring function for our phrase to be automatically be assigned to our extracted topics will lead to a better understanding of the thematic subject of each topic. The fact that the topic extraction and phrase generation will use the same dataset will result to better tuned results.

4.2. Text and Data Mining

Text mining or text data mining is the process of deriving high quality information from text. This high quality information is typically derived through the devising of patterns and trends through means such as statistical pattern learning. Text mining usually involves the process of structuring the input text, deriving patterns within the structure data and finally evaluation and interpretation of the output. The high quality is data usually refers to how interesting or relevant the results are. Typical text data mining includes text categorization, text clustering, sentiment analysis, concept extraction, relation modeling and document summarization. Text analysis involves information retrieval, word frequency distributions, pattern recognition, word tagging, information extraction, visualization and predictive analytics.[13] The goal of text data mining is to turn our text into data ready for analysis. Text mining is widely used for social media monitoring and social media mining. Mining the social media has its potential to extract actionable patterns that can be beneficial for users, businesses and costumers.

In order to do text mining we need to develop a method for the representation of our documents. It is sufficient for our kind of data to classify and cluster our documents using simple representation that loses all information about word order (bag of words). Given our collection of documents, the first task is to identify the set of all words used in our documents. Using our data trimming techniques that we developed will be useful for extracting categories in our documents. We aim at representing our documents as a two-dimensional matrix where each row describes a document and each column corresponds to a word.

4.2.1 Topic modeling

Once we have a representation of our documents we need to select a model for a set of documents. The model will be an abstraction of a set of entities using probability distribution. Given a training set of documents we will choose values for the parameters of a probabilistic model that make the training documents have high probability. By throwing test documents to our trained documents we will be able to evaluate its probability according to the model. The higher the probability the more similar to the training set. The probability distribution is going to be multinomial and can be represented in the following mathematical formula:

$$p(x;\theta) = \left(\frac{n!}{\prod_{j=1}^{m} x_j!} \right) \left(\prod_{j=1}^{m} \theta_j^{x_j} \right).$$

Where x is a vector of non-negative integers and the parameters θ are a real-valued vector. Both vectors have the same length m . θ_j is the probability of word j while x_j is the count of word j . [15]

4.2.2 Generative processes

A common way to organize a collection of documents is to use unsupervised learning techniques. The generative process is a specification of a parameterized family of distributions. Learning is based on the principle of maximum likelihood or maximum probability. [16]

A generative process for a simple document is as follows:

- Fix a multinomial distribution with parameter vector ϕ of length V (setting up the probability distributions)
- For each word in a document: draw a word w according to ϕ

For a collection of documents of multiple categories, a simple generative process is:

- Fix a multinomial α over categories 1 to K. For category number 1 to category number K: Fix a multinomial with parameter vector ϕ_k
- For a document number 1 to document number M: Draw a category z according to a and for each word in the document: Draw a word w according to ϕ_z .

4.2.3 Latent Dirichlet allocation (LDA)

The specific topic model that we consider for our research is called Latent Dirichlet Allocation (LDA). Latentt Dirichlet Allocation is arguable the most popular and simplest topic model in application. It has also been widely used for topic modeling in micro-blogging data providing accurate and meaningful results. Latent Dirichlet Allocation is a generative model that allows sets of observations to be explained by unobserved groups that explain why some parts of data are similar. In Latent Dirichlet Allocation each document may be viewed as a mixture of various topics and topic distribution is assumed to have a Dirichlet prior. Latent Dirichlet allocation is based on the intuition that each document contains words from multiple topics where the proportion of each topic in each document is different, but the topics themselves are the same for all documents.[16] The generative process assumed by the Latent Dirichlet Allocation is as follows:

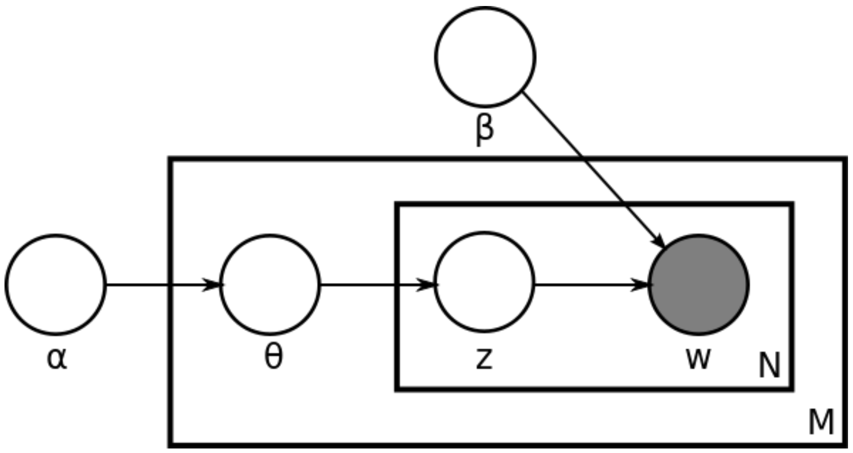


Figure 7: Latent Dirichlet allocation representation model

- a is the parameter of the Dirichlet prior on the per-document topic distributions
- β is the parameter of the DIrichlet prior on the per-topic distribution
- θ_i is the topic distribution for the document i
- ϕ_k is the word distribution for topic k
- z_{ij} is the topic for the j^{th} word in document i
- w_{ij} is the specific word

Latent Dirichlet Allocation uses Dirichlet distribution. Dirichlet distribution is a probability density function over the set of all multinomial parameter vectors. This set is all vectors γ of length m such that $\gamma_s \geq 0$ for all s and $\sum_{s=1}^{m} \gamma_s = 1$. The Dirichlet distribution itself has a parameter vector a of length m and the equation is:

$$p(\gamma \mid \alpha) = \frac{1}{D} \prod_{s=1}^{m} \gamma_s^{a_s - 1}$$

Where the function D is a normalized constant.

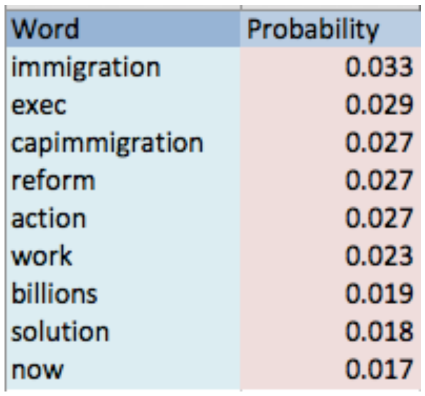
4.2.4 Topic Discovery and Classification

Using Latent Dirichlet Allocation with our micro-blogging data may not conclude to quality topic extraction due to the fact that tweets are short. It has been suggested by previous researchers that by treating tweets as a document might result into better solutions however we have been able to extract semantically meaningful topics in both ways. Python's Gensim library for topic modeling will provide us with the ability to apply Latent Dirichlet Allocation algorithm for topic discovery in our dataset. Again using a small toy example dataset that we have created by tracking economy related tweets using the following code can extract n-number of topics.

```
stoplist=set(output.read().split())
texts = [[word for word in tweet.lower().split() if word not in stoplist]
        for tweet in tweetList]

# # remove words that appear only once
all_tokens = sum(texts, [])
tokens_once = set(word for word in set(all_tokens) if
all_tokens.count(word)== 1)

texts = [[word for word in text if word not in tokens_once]
        for text in texts]

dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]
lda = ldamodel.LdaModel(corpus, id2word=dictionary, num_topics=6)
corpus_lda = lda[corpus]
```

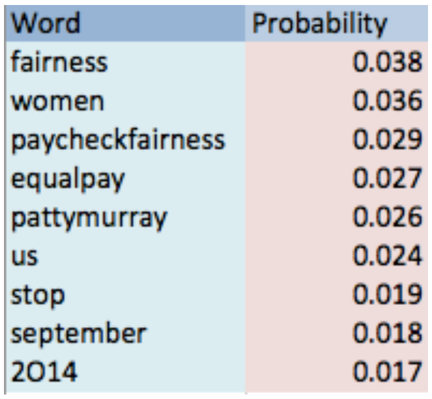
```
for l,t in izip(corpus_lda,corpus):
        print l,"::",t

for i in range(0,lda.num_topics):
        print lda.print_topic(i)
        topics.append(lda.print_topic(i))
```

Using again stop words removal and link removal in our dataset we create then a dictionary of our corpus and using this dictionary to generate the latent Dirichlet allocation model for our dataset using 6 number of topics. The results can be represented below displaying 3 of the best topics extracted as bags of words with a probability distribution for every word:



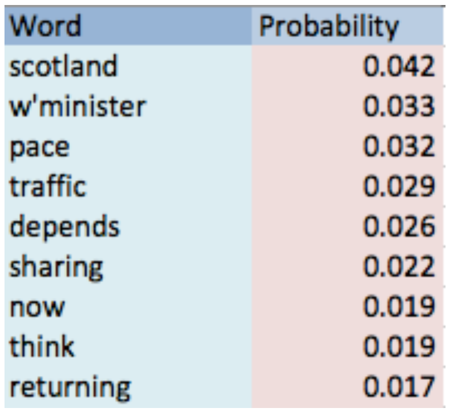
Word	Probability
immigration	0.033
exec	0.029
capimmigration	0.027
reform	0.027
action	0.027
work	0.023
billions	0.019
solution	0.018
now	0.017

Table 1: Topic 1 extracted



Word	Probability
fairness	0.038
women	0.036
paycheckfairness	0.029
equalpay	0.027
pattymurray	0.026
us	0.024
stop	0.019
september	0.018
2014	0.017

Table 2: Topic 2 extracted



Word	Probability
scotland	0.042
w'minister	0.033
pace	0.032
traffic	0.029
depends	0.026
sharing	0.022
now	0.019
think	0.019
returning	0.017

Table 3: Topic 5 extracted

We can easily see that the topics 1, 2, 5 extracted from our database can relate to real word thematic categories regarding economy as a general subject. Topic 1 is related with tweets in our dataset that talk about immigration and how affects the economy. Topic 2 is related with the rights of women to be paid equally as the men in their workplace. Topic 5 is related with Scotland dependency from United Kingdom and how it affects economy.

4.2.5 Probabilistic Topic Labeling

Deriving meaning of labeling a topic manually might not be so easy as it was from our previous examples. Most of the times it would be impossible for a human to interpret the information from extracted topic in order to label it correctly. To generate labels that are understandable and semantically relevant across our topics as we explained in our problem definition we need to generate automatically meaningful phrases as candidate labels. We already described our methodology for phrase generation from our dataset using natural language processing techniques such as chunking/shallow parsing. Having already generated a big number of candidate labels for our topics (phrases) from our dataset we now need to develop a scoring function in order to assign the highest scoring phrase to each topic.

The semantics of a latent topic θ is fully captured by the corresponding multinomial distribution. Any reasonable measure of the semantic relevance of a label to a topic should compare the label with this distribution in some way. [2]

We are going to define the semantic relevance score of a candidate phrase $l = u_0 u_1 .. u_m$ where u_i is a word as follows:

$$S = \log \frac{p(l \mid \theta)}{p(l)} = \sum_{0 \le i \le m} \log \frac{p(u_i \mid \theta)}{p(u_i)}$$

The basic idea behind this *zero-order relevance scoring function* is that a phrase that contains more “important” words in the topic distribution is assumed to be a good label. $p(u_i)$ is to correct the bias toward favoring short phrases and we are choosing to set it to uniform distribution. Using this function now we can score every generated phrase from our chunker and based on the highest scored phrase for each topic we can assign these labels (phrases) to each topic θ. An illustration of zero-order relevance is demonstrated below. Larger circle infers to a higher probability value.

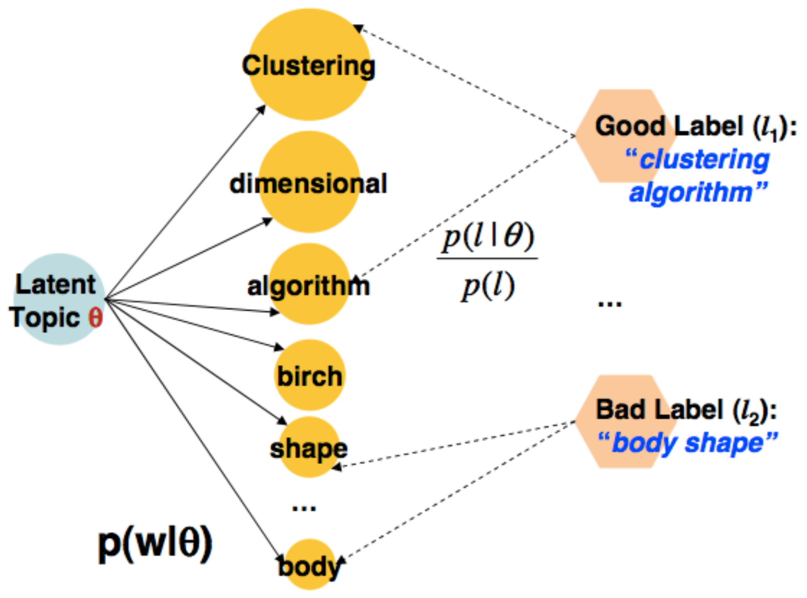


Figure 8: Zero-order relevance scoring function

The following Python script illustrates the implementation of the scoring function by storing a scoring array with all the scores of each phrase for every topic separately:

```
for topic in topics:
        print 'New Topic:------------------------------'
        with open('phraseschunker.csv','rU') as csvfile:
        phrases = csv.reader(csvfile, delimiter=' ',quotechar='|')

        for phrase in phrases:
          joinphrase=' '.join(phrase)
          b=topic.split('+')
          for item in b:
          a=item.split('*')
          for p in joinphrase.split():
            print p,a[1]
            if str(p) in str(a[1]):

              print  'Probability', a[0]
              print 'Phrase:', phrase
              print 'Topic:' ,topic
              score=float(math.log10(float(a[0])+ float(1)))+score
        scoretable[m].append(score)
        scpre=0
```

In the following Python script we find the maximum score in the scoretable array for each topic and assign the phrase to the topic as a label:

```
for i in range(len(topics)):

        maxphrase[i] = phrzs[scoretable[i].index(max(scoretable[i]))]

        print max(scoretable[i]),scoretable[i].index(max(scoretable[i]))
         print phrzs[scoretable[i].index(max(scoretable[i]))]
        print lda.print_topic(i)
```

4.3 Social Network Analysis

In order to translate our extracted information into a social network with nodes and edges that carries significant information and insights about the community structures, the most influential nodes (users) in the network and the trends that dominate each community we need to implement graph theory methods, apply community detection algorithms and provide a suitable environment for visualization of this network and its information. This way we can represent our network using graphs but we will need to develop a method in order to generate our graph using our dataset. This task involves a precise definition of nodes, edges (interactions) and communities (entities) in our dataset.

4.3.1 Graph Generation

Graphs contain both a set of objects, called nodes, and the connections between these nodes called edges. Mathematically, a graph G is denoted as pair $G(V,E)$, where $V=\{v_1,v_2,...,v_n\}$ with nodes v_i, for $1\leq i\leq n$ and represents the set of nodes and $E=\{e_1,e_2,...e_m\}$ represents the set of edges with e_i, for $1\leq i\leq m$ being the edges for our nodes.

Nodes: it is obvious that the nodes in our graph are going to represent each Twitter user that has tweeted and we had collected his tweet using our Twitter wrapper (tweepy).

Edges: As edges or interactions between our nodes (users) in our graph we could define them by assigning them into followers/following interactions. Despite that this definition can create interesting graphs of Ego-networks, it is not going to provide us with interesting patterns in terms of trends or community discovery. Friendship-like connection does not necessarily mean that these friendship communities have the same topic interests as individuals and this is why we need to develop another strategy/method for creating interactions in our graph. We are interested in “ad-hoc” connections between users generated by current trends or topic interests in their community and how discussion development among them is sustained. By analyzing the format of a tweet we conclude that “mentions” of a node (user) can represent the ideal definition for an edge in our network. A mention is a mean of posting references or links to a user’s profile. The following example will demonstrate the edge generation between three nodes (users) posting three tweets:

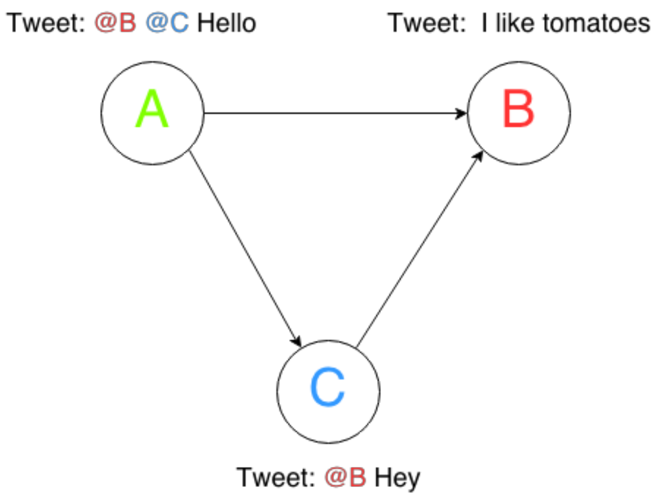


Figure 9: Demonstration of edges in a small network

Using these definitions for nodes and edges in our network we can develop a method to create a graph using NetworkX Python’s library which is widely used for graph related problems and build our network based on the data we gathered using our Twitter wrapper and stored to MongoDB. The data we are going to need from a tweet’s JSON data are going to be:

```
{
"user":
{ "screen_name": { "name of the user"}}
"text": "This is the actual tweet of the user",
"entities": {
        "user_mentions":
                  [{      "screen_name": "username of the user that the node
                                      is mentioning in his tweet",     }]   }
```

```
 "geo": {              "coordinates": { "x y z" }
 "created_at": { "to get the information about the time that the tweet has
been created"}
"source":{ "to extract the information about the hardware or software that
the user is using"}
 }
```

Using MongoDB as our database system we don't need to use regular expression to modify the information that Twitter API provides us because the data of a tweet is already formatted in JSON format and MongoDB represents data in JSON format too. We have developed the following script in Python in order to retrieve this data from our Twitter API and store it to a MongoDB database in our system:

```
Class StreamListener(tweepy.StreamListener):
    def __init__(self, api):
       self.api = api
       super(tweepy.StreamListener, self).__init__()
       self.db = pymongo.MongoClient().inflationDB

    def on_status(self, status):
       print status.text
       data ={}
       data['user']=status.user.screen_name
       data['text'] = status.text
       data['created_at'] = status.created_at
       data['geo'] = status.geo
       data['source'] = status.source
       data['entities']=status.entities

       self.db.Tweets.insert(data)
```

After the data gathering we can implement our Python script that utilizes NetworkX library in order to generate our graph. We access every user and add each user name as a node and then we scan the screen name of this user mentions in order to create our edges:

```
#connecting with our database
c=Connection()
db=c.EbolaNet  #my database #Another DB: EconomicsDB,newEbolaDB
tweets=db.Tweets
 #creating the empty graph for networkX that we are going to populate
mynetwork=nx.Graph()
mentions=db.Tweets.entities.user_mentions tweets=tweets.find()
test=tweets.entities.user_mentions.find()
```

```
for tweet in tweets:
    mynetwork.add_node(t['user'],tweet=t['text'])

    if len(t['entities']['user_mentions'])!=0:
        try:
           While(i<6):
               mynetwork.add_edge(t['user'],t['entities']['user_mentions'
                                                  ][i]['screen_name'])
               i=i+1
        except:
            pass
```

4.3.2 Community Discovery

A real-world community is a body of individuals with common interests. A virtual community comes into existence when users with common interests start interacting with each other. The problem of community detection has been tackled and discussed by many different disciplines such as quantization in electrical engineering, discretization in statistics and clustering in machine learning. We want to extract communities that share common thematic interests. As we stated in our problem definition chapter, the most efficient way to implement community detection in our network (graph) is by using Louvain method that is a modularity optimization technique in order to detect communities in large networks. This community detection algorithm is divided in two phases that are repeated iteratively. At first it assigns a different community to each node of the network. This way in this initial partition there are as many communities as there are nodes. Then, for each node i this algorithm is assigning the neighbors j of i and evaluates the gain of modularity that would take place by removing i from its community and by placing it in the community of j. The node i then placed in the community for which this gain is maximum (only if the gain is positive). This process is applied repeatedly for all the nodes in our network until no further improvements can be made. A Python module that uses Louvain method in NetworkX's framework is called. We are going to implement this module into our graph in order to extract communities. This implementation can be demonstrated in the code below continuing from the previous script as we have already generated our graph and it creates a list of nodes for each community:

```
import networkx as nx
import community
```

```
#first compute the best partition
partition = community.best_partition(mynetwork)
size = float(len(set(partition.values())))
pos = nx.spring_layout(mynetwork)
count = 0
# print partition.values()
for community in set(partition.values()) :
    count = count + 1
    list_nodes = [nodes for nodes in partition.keys() if partition[nodes] ==
community]
    #random colors in every loop
    t='#'+str(hex(random.randint(0,16777216))[2:])
    for node in list_nodes:
        nodesnameslist.append(node)
```

4.3.3 Community Labeling

Now that we have been able to detect communities in our graph we want to be able to identify influential users (nodes) or the topic (or topics) of interest in each community. In order to do that we need to define a method in order to calculate how influential a user is in a community and also to be able to retrieve the information about all the nodes in each community. In order to proceed we need to implement again some techniques from graph theory.

4.3.3.1 Degree

In graph theory, the degree of a vertex of a graph is the number of edges incident to the vertex. The degree of a vertex v is denoted $\deg(v)$.

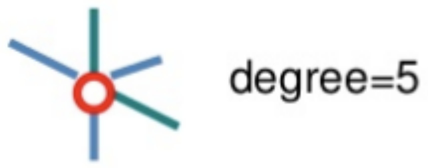


Figure 10: Node A with degree=5

In directed graphs like our network the set of nodes are connected by directional edges. When a user mentions another user in his tweet, this interaction produces a directed edge from user

A to user B. Indegree is the number of directed edges (arcs) are incedent on a node and the outdegree is how any directed edges (arcs) originate at a node.

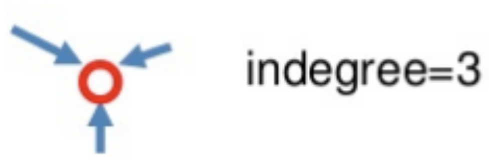


Figure 11: Node indegree

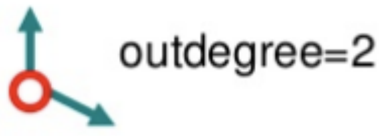


Figure 12: Node outdegree

4.3.3.2 Influence

Centrality in graph theory is a term that defines how important a node is within a network. In real-world interactions, we often consider people with many connections to be important. Degree centrality transfers the same idea into a measure. The degree centrality measure ranks nodes with more connections higher in terms of centrality. The degree centrality C_d for node v_i is: $C_d(v_i) = d_i$, where d_i is the degree of node v_i. [17]

This way we can compute the degree of each node in every community in our graph using netowrkX's functionality and store it in a list.

4.3.4 Community Analysis

Now that we are able to access every node in each community we can retrieve each tweet from our nodes and measure the most relevant topic of interest of each community by re-using the zero-order relevance scoring function that we have developed as a phrase candidate generation mechanism previously, modified for a tweet and a topic. Given two extracted topics θ_1, θ_2 and a tweet, t in a community, using our scoring function $S(\theta_i, t)$ we could calculate the relevance score of a topic θ_i for a tweet θ_i. Assuming that $S(\theta_{1,}t) > S(\theta_2, t)$ suggests as that topic θ_1 is most relevant to tweet t and therefore topic θ_1 is going to represent tweet t. Following the same procedure for every tweet in a community for each community we can result with a table of representative topics for each topic. Then we can calculate the percentage of existence for each community. In our scoring function we are going to use again LDA's probability distribution as dependent probabilities in order to provide a more accurate representation meaning that when we are applying the function each word in a tweet is going to be compared with each word in the topic applying the coefficient of the probability for this word to appear in the topic context. This procedure can be represented by the following figure:

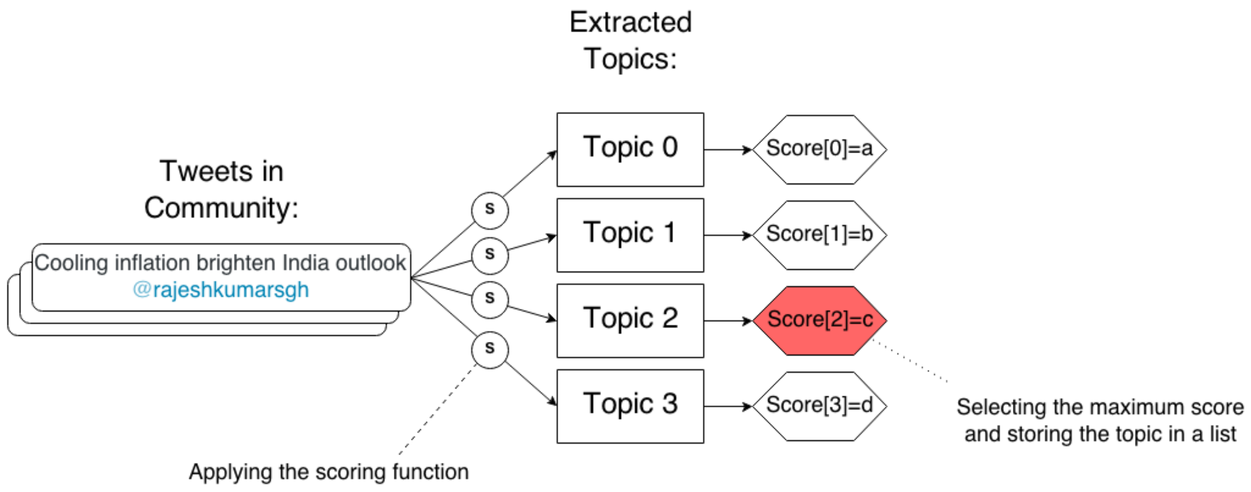

Extracted Topics:
Tweets in Community:
Cooling inflation brighten India outlook
@rajeshkumarsgh
s
Topic 0
Topic 1
Topic 2
Topic 3
Score[0]=a
Score[1]=b
Score[2]=c
Score[3]=d
Selecting the maximum score and storing the topic in a list
Applying the scoring function

Figure 13: Scoring process

This procedure is performed for each node in every community that we have been able to detect. This way we can calculate the most relevant topic of interest in every community with a percentage of relevance. This can be implemented in the code below (we have not included the whole code).

```
if (len(list_nodes)>5):
    for node in list_nodes:
        nodesdegreelist.append(mynetwork.degree(node))
        nodesnameslist.append(node)
        try:
            d=mynetwork.node[node]['tweet']
            tokens=nltk.word_tokenize(d)
            filteredtext = [t for t in tokens if t.lower() not in stopList]

            tokenizedtweet=' '.join(filteredtext)
            print 'TWEET------------------------------',tokenizedtweet

            st=[]
            for topic in topcs:
                for word in tokenizedtweet.split():

                b=topic.split('+')
                for item in b:
                    a=item.split('*')
                    if word in a[1]:
                        score=float(math.log10(float(a[0])+float(1)))+score
```

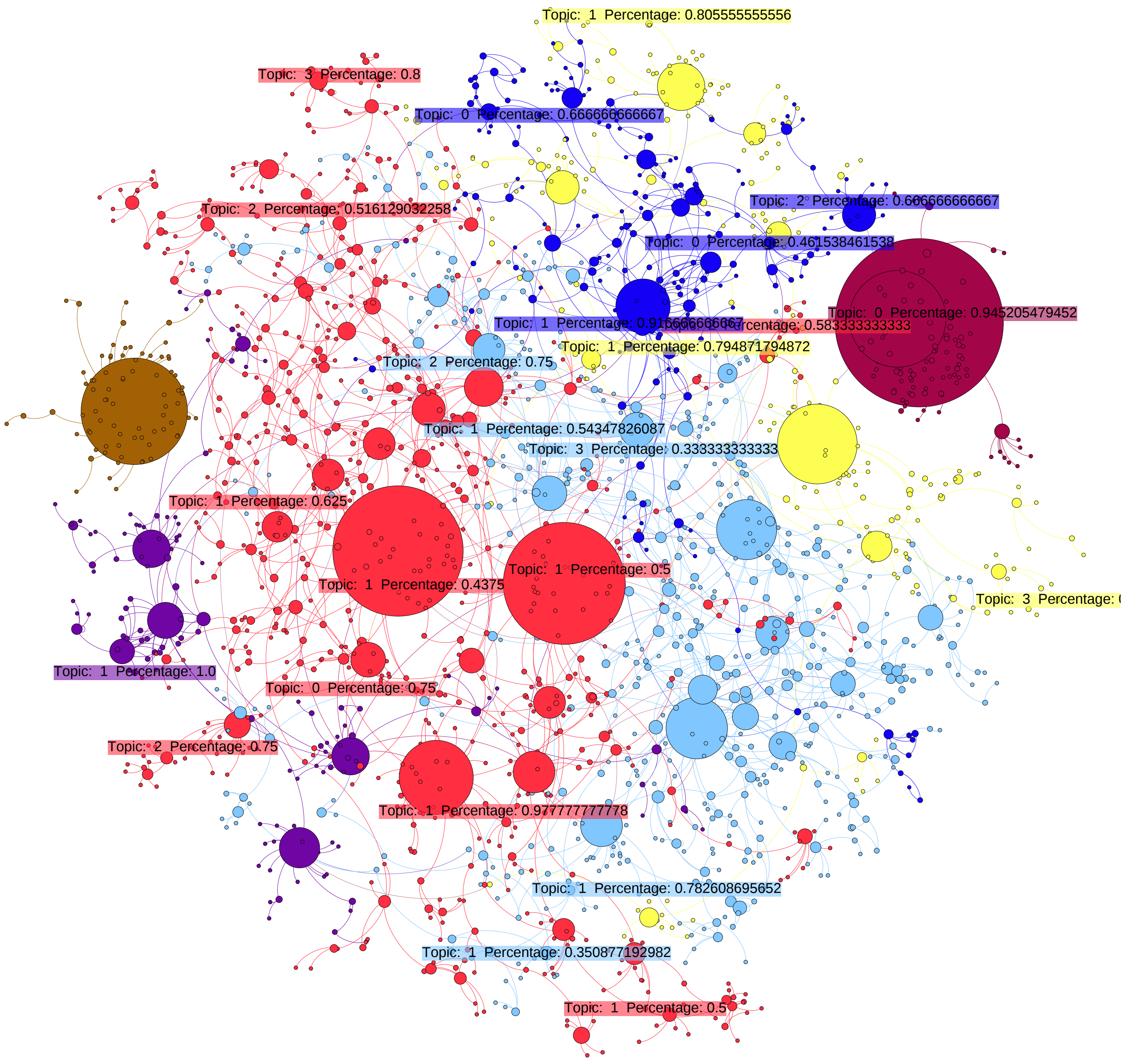
```
                  st.append(score)
                  score=0.0
         except:
                 pass

         if(max(st)>0):
             print st.index(max(st))
             communitytopicscores.append(st.index(max(st)))

     if (len(communitytopicscores)>0):
         for i in range(len(num_topics)):
             print communitytopicscores.count(i)
             topiccounts.append(communitytopicscores.count(i))
             #computing the percentage of the most relevant topic
             perc=max(topiccounts)/len(communitytopicscores)

             topicassignment='Topic:
'+str(topiccounts.index(max(topiccounts)))+'  Percentage: '+str(perc)

             mynetwork.add_node(node,topic=topicassignment)

     communitytopicscores=[]
     topiccounts=[]
     nodesdegreelist=[]
     nodesnameslist=[]
```

Using this script we are calculating the scores of each topic for every tweet's node separately and we are keeping tha maximum topic score value and append these scores to a list `st`. After the `except` function in our Python script, we append the maximum scored topic's index number to another list `communitytopicscores` with the communities topic scores. Next we are computing the percentage of relevance between the community and the topic in a float variable `perc` and we assign this variable's value in the string variable `topicassignment`. We are using NetworkX's functionality to add the node with the max degree in our network with a label that carries information about the community that this node represents. This information is the most relevant topic to the community and the percentage of this relevance. Adding a node in a network that already consists this node will just adjust any additional information that this nodes carries. In our the node carries the label of `topicassignment` variable. Using this method we can provide useful insights about the communities in our networks but we can also help with the visualization by providing an improved aesthetically representation of our network by not exhausting the image of our network with information in every node without allowing the patterns to be visible. Using this algorithm the output is looking like the figure below for some communities:

```
Most influential (highest degree) node in the community: NBCNews

Most relevant topic in his community: Topic 1 with percentage of
0.916666666667

-----------------------------------------------------------------------------

Most influential (highest degree) node in the community: Earthjustice

Most relevant topic in his community: Topic 0 with percentage of
0.7247667422

-----------------------------------------------------------------------------
```

4.3.5 Network Visualization

Regarding visualization aesthetics it could help us to adjust the size of each node in our network depending on the degree. This way we can identify important and influential nodes quickly in a large network. We also need to find a suitable layout algorithm for the representation of our network because the layout presets of NetworkX are not scalable for large networks and NetworkX does not allow quick adjustments in our graphs. We are going to use Gephi, which is an interactive visualization and exploration platform for all kinds of netowrks and complex systems. As we already suggested in the previous chapter, using a force-directed layout algorithm for real world large-scale graphs is suggested. OpenOrd C++ [24] implementation is an algorithm based on Fritcherman-Reingold and works with a fixed number of iterations. The algorithm is using simulated annealing and has five phases: liquid, expansion, cool-down, crunch and simmer. We can also adjust the size of each node depending on the degree using Gephi. Applying modularity classification to our already processed network can also create different colors for every partition (community) and this way we can filter out small communities that does not obey a centrality threshold. An example of a network with 9699 nodes and 6535 edges that has been generated by tracking tweets about Ebola virus can be shown below:


Topic: 1 Percentage: 0.805555555556
Topic: 3 Percentage: 0.8
Topic: 0 Percentage: 0.666666666667
Topic: 2 Percentage: 0.666666666667
Topic: 2 Percentage: 0.516129032258
Topic: 0 Percentage: 0.461538461538
Topic: 0 Percentage: 0.945205479452
Topic: 1 Percentage: 0.916666666667
ercentage: 0.583333333333
Topic: 1 Percentage: 0.794871794872
Topic: 2 Percentage: 0.75
Topic: 1 Percentage: 0.54347826087
Topic: 3 Percentage: 0.333333333333
Topic: 1 Percentage: 0.625
Topic: 1 Percentage: 0.4375
Topic: 1 Percentage: 0.5
Topic: 3 Percentage: (
Topic: 1 Percentage: 1.0
Topic: 0 Percentage: 0.75
Topic: 2 Percentage: 0.75
Topic: 1 Percentage: 0.977777777778
Topic: 1 Percentage: 0.782608695652
Topic: 1 Percentage: 0.350877192982
Topic: 1 Percentage: 0.5

5. Experiments and Results

In this chapter we are going to examine our results analytically providing empirical evaluation.

Datasets: We are going to use different datasets that we have created using Twitter's API. Our datasets are: an economy related dataset that includes 3161 tweets, another economy dataset that includes 7135 tweets and a technology dataset that includes 10802 tweets.

Procedure: The procedure that we are going to follow using the software modules that take advantage of the methodologies that we have developed can be shown in the diagram below.

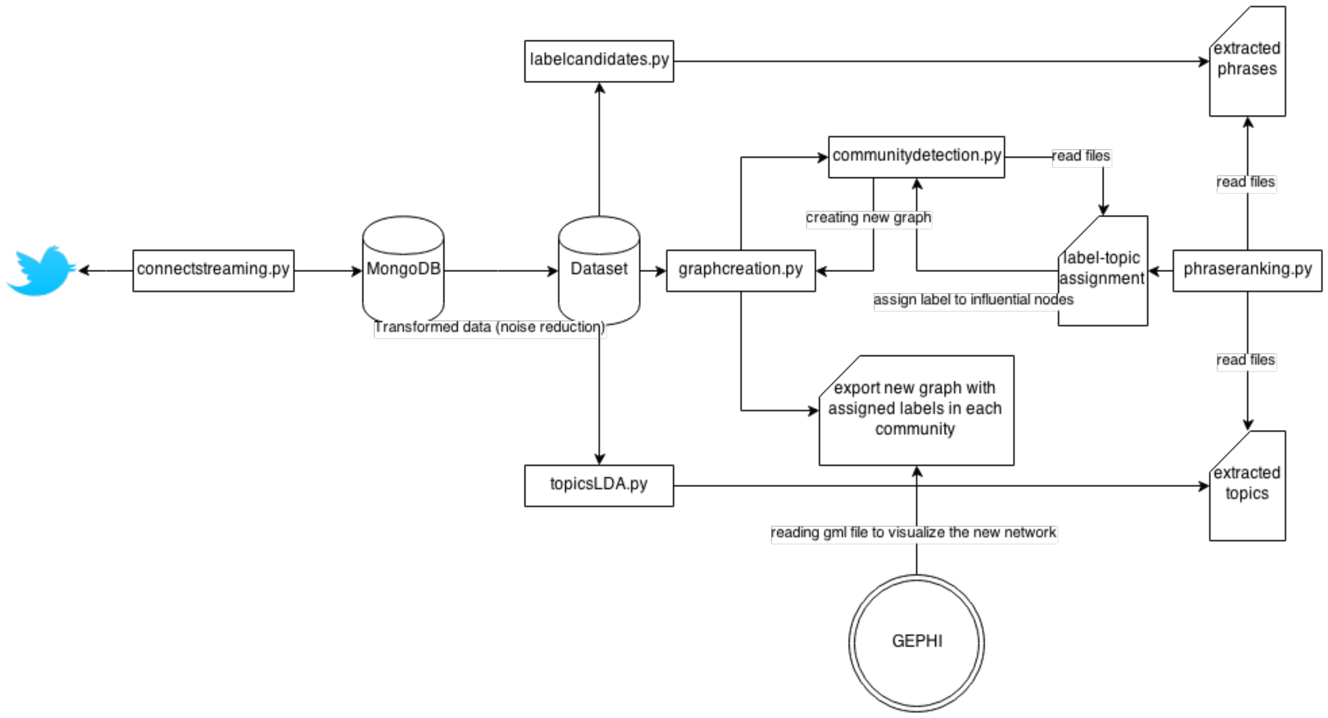


Figure 14: Procedure diagram

5.1 Phrase Generation

In this section we are going to present the effectiveness of the chunking/shallow parsing method that we have proposed in order to extract grammatically correct phrases from our text corpus (tweets) using two different datasets and three different chunking methods.

The top extracted phrases using our datasets that we had created in MongoDB containing tweets that are related with economy and technology are going to be presented below using the three phrase patterns that we have developed in our methodology:

- Phrase Pattern A: `{<RB.?>*<VB.?>+<NNP}`
- Phrase Pattern B: `{<JJ\w?|VG\w?|NN\w?|DT\w?>*<NNP>+}`
- Phrase Pattern C: `{<NN\w?>*<DT\w?|NN\w?>*<JJ\w?|VG\w?>+<NN\w?>+<JJ\w?|VG\w?>*}`

Some tweets in our economy dataset that our *chunkers* are going to use are:

Economy Dataset example
OECD sees global economy held back by slow eurozone http:\/\/t.co\/BHNzJOcGYH
RT @SCV_Network: Lets help support our local businesses and buy local! Buy local, impove local economy.
RT @shoplocally: It's not hard to support your local economy. Just shift your spending to local independents. Every bit counts. #ShopLocal
Growing Our Economy and Strengthening Our Financial System | The White House http:\/\/t.co\/guNt4fimp
@Bruciebabe Consumer driven Low wage economy
RT @FlipChartRick: The Scottish economy in ten essential charts | via @Telegraph http:\/\/t.co\/Km2bzXos0

Table 4: Economy Dataset example

The extracted phrases using our three chunking methods after the noise reduction are:

Phrase Pattern A
creating jobs
improve economy
transform u.s.
reduce deficit
fighting isis
collapsed economy
nearly fighting isis
hurting u.s.
economy tanking
giving u.s.
collapsing system
recovered economy
helping st.louis
drone economy
boost economy

Table 5: Extracted phrases

Phrase Pattern B
local economy
local independents
unpaid women
fairness
local news
congressional puttymurray
undervalued jobs
great news
lowest wage
lower wage
5-year low
expensive states
new digital-based
scotish
digital economy

Table 6: Extracted phrases

Phrase Pattern C
global economy
local economy
local independents
financial system
bad news
checked bags
small banks
low wage economy
bad economy
small economy business
the lowest benefit
new tax laws
financial crisis
the scottish economy
scottish sector

Table 7: Extracted phrases

The tweets in our technology dataset can be shown below:

Technology Dataset example
#Apple - Latest rumors before Tuesday's Apple event: #iPhone \/ #iPhone6 - The new phone
#Technology - Cyborg Unplug scans your Wi-Fi network for potential surveillance devices: #Wifi... http:\/\/t.co\/9TuZRfCIb4 - #Tech #Techno
why some healthcare technology leaders have been hesitant to embrace cloud-based technology writ large\" http:\/\/t.co\/oOI1bktqAs
#Google - Sony Xperia T3 review: A solid mid-range large-screen phone: #XperiaT3 - Sony's ... http:\/\/t.co\/msZiM5qyrq - #Tech #Technology
How Mobile Technology Can Help Grow Your Tutoring Agency via @edukwest\n\nhttp:\/\/t.co\/Dh0ZkRL7e
The Solar Technology Behind Apple's iPhone 6 http:\/\/t.co\/gFAfZjhx0c

Table 8: Technology Dataset example

The extracted phrases from our technology dataset after the noise reduction using our three chunking methods are:

Phrase Pattern A
apple iphone6
outdated technology
rebel t5i
folding laundry
regulate e-rickshaws
switzerland p2p
alarming co2
relive itswc14
now hiring
deserted manhatan
print technology
refurbished w0g02289
breaking news
inventing apps
selling games

Table 9: Extracted phrases

Phrase Pattern B
new iphone6
new technology
futuretech amnc14
new zealand
low-cost technology
best calculator
huge news
operational costs
atmospheric co2
new iphone
human body
least important thing
greatest delivery
neu app
bad technology

Table 10: Extracted phrases

Phrase Pattern C
apple latests rumors
potencial surveillance devices
cloud technology
four-year-old processor smartwatch
powerful passwords
healthcare infographic technology
large-screen phone xperiat3
missile technology
educational blogs
vocational training
big data technology
pittsburgh technology
mobile technology
next big breakthrough
solar technology

Table 11: Extracted phrases

As we can see, the phrases extracted with the chunking method,

Phrase Pattern C:

```
{<NN\w?>*<DT\w?|NN\w?>*<JJ\w?|VG\w?>+<NN\w?>+<JJ\w?|VG\w?>*}
```

most of the time this method for phrase generation provide meaningful phrases that can capture more accurately the thematic domain of our dataset in contrast with Phrase Pattern A and Phrase Pattern B which are producing mediocre results and their extracted phrases are more abstract which can lead to misleading interpretation of a tweet or a topic. Therefore it is more likely for Phrase Pattern C type of phrases to become candidates for topic labels.

5.2 Topic Extraction using LDA

As we already analyzed in chapter 4, we are going to use Python's Gensim library to implement Latent Dirichlet Allocation algorithm to our datasets to extract topics. Testing LDA in a relatively small dataset (3161 tweets) related to economy we are going to use the following initializations for our LDA algorithm:

```
lda = ldamodel.LdaModel(corpus, id2word=dictionary, num_topics=6)
```

which uses the default presets of LDA model, where `id2word` is the mapping from words ids (integers) to words (strings) and `num_topics` is the number of the requested topics. The results are going to be presented in tables where T_i for $i \geq 1$ and $i \leq num\_topics$, and p will be the probability of the word distributed in the document:

T1	p	T2	p	T3	p	T4	p	T5	p	T6	p
trouble	0.030	add	0.026	economy	0.023	financial	0.028	w'minister	0.036	lead	0.021
local	0.023	exec	0.026	pattymurray	0.023	people	0.021	now	0.031	financial	0.021
jobs	0.022	immigration	0.026	equalpay	0.023	world	0.021	sharing	0.025	economy	0.014
2014	0.022	cappimigration	0.025	women	0.023	reform	0.021	til	0.019	25	0.014
deep	0.022	action	0.025	countries	0.023	undervalued	0.015	depends	0.019	us	0.014
low	0.022	billions	0.025	fairness	0.022	cegx	0.015	souring	0.019	checked	0.014
september	0.018	action	0.025	ik	0.022	cheap	0.015	things	0.019	improve	0.014
food	0.015	reform	0.020	paycheck	0.020	way	0.014	scotland	0.014	westjet	0.014
economy	0.015	lead	0.020	voor	0.020	2014	0.014	wait	0.014	arab	0.013
banks	0.014	tories	0.019	now	0.017	em	0.014	lead	0.013	system	0.012

Table 12: Extracted topics from economy dataset

As we can see, using our economy dataset with default settings for our LDA model the six extracted topics are formed into patterns that have a semantic meaning, with some minor abstractions. Now we are going to experiment with the LDA model parameters in order to try to export better results.

```
lda = LdaModel(corpus, num_topics=6, alpha='auto', eval_every=5)
```

Using these settings we are going to use `eval_every` parameter, which slows down the training of the model providing better accuracy but less performance (the default is 10).

Alpha parameter affects the sparsity of the document-topic (theta) and topic-word (lamda) distributions. Setting alpha parameter to ‘auto’ our model is going to learn an asymmetric prior directly from our data. The results can be shown in the table below:

T1	p	T2	p	T3	p	T4	p	T5	p	T6	p
local	0.035	immigration	0.022	women	0.026	bjp	0.017	uk	0.027	us	0.034
deep	0.020	pace	0.020	2014	0.026	inflation	0.017	scotland	0.027	economy	0.028
trouble	0.020	2014	0.020	economy	0.018	3.74	0.017	think	0.020	vote	0.023
support	0.015	w’minister	0.015	equalpay	0.018	silence	0.017	sharing	0.020	god	0.023
shoplocal	0.015	action	0.015	fairness	0.018	low	0.016	lead	0.020	2014	0.023
shift	0.015	reform	0.015	tax	0.018	deafening	0.016	depends	0.014	action	0.022
spending	0.015	billions	0.014	paycheck	0.018	minhazmerchant	0.016	hugorifkind	0.014	crisis	0.022
hard	0.015	add	0.014	people	0.018	5-year	0.015	now	0.013	now	0.012
economy	0.014	fwd_us	0.013	firms	0.018	cong	0.015	w’minister	0.013	financial	0.012
shoplocally	0.014	cappimigration	0.013	new	0.017	falls	0.015	wait	0.013	state	0.012

Table 13: Extracted topics from our economy dataset (new parameters)

As we can see, the results has not changed radically however now we can definitely determine each inter-topic’s category manually.

- Topic 1: is related with local economy and issues related with not supporting local shops.
- Topic 2: is related with immigration and how it affects economy
- Topic 3: is related with woman rights
- Topic 4: is related with inflation, and probably Bharatiya Janata Party is aiming to defeat inflation
- Topic 5: is related with Scotland independence and how it will affect the economy
- Topic 6: is related with the economy of United States, elections and the financial crisis

The next step will be to experiment with the results of topic labeling using phrases.

5.3 Topic Labeling using Zero-Order Scoring Function:

In this section we are going to present the results and the effectiveness of the proposed method for automatically labeling topic models using our datasets.

T1	p	T2	p	T3	p	T4	p	T5	p	T6	p
local	0.035	immigration	0.022	women	0.026	bjp	0.017	uk	0.027	us	0.034
deep	0.020	pace	0.020	2014	0.026	inflation	0.017	scotland	0.027	economy	0.028
trouble	0.020	2014	0.020	economy	0.018	3.74	0.017	think	0.020	vote	0.023
support	0.015	w'minister	0.015	equalpay	0.018	silence	0.017	sharing	0.020	god	0.023
shoplocal	0.015	action	0.015	fairness	0.018	low	0.016	lead	0.020	2014	0.023
shift	0.015	reform	0.015	tax	0.018	deafening	0.016	depends	0.014	action	0.022
spending	0.015	billions	0.014	paycheck	0.018	minhazmerchant	0.016	scottish	0.014	crisis	0.022
hard	0.015	add	0.014	people	0.018	5-year	0.015	now	0.013	now	0.012
economy	0.014	us	0.013	firms	0.018	poor	0.015	w'minister	0.013	financial	0.012
shoplocally	0.014	cappimigration	0.013	new	0.017	falls	0.015	wait	0.013	state	0.012

Table 14: Extracted topics from our economy dataset for phrase ranking

The top labels for each topic are:

- $label_{T1}$: local economy, score= 0.042055161736
- $label_{T2}$: us lowest pace, score=0.030886526429
- $label_{T3}$: paycheck fairness, score=0.030912453133
- $label_{T4}$: poor economy, score=0.018120597038
- $label_{T5}$: scottish economy, score=0.017492753773
- $label_{T6}$: economy financial crisis, score=0.055814264886

As we can see, our extracted phrases can capture the essence of every topic despite that the lower scores below become abstract or a misleading in some cases. The zero-order function generates quality labels for our topics. Now we are going to extract the labels for our topics extracted from our technology dataset.

T1	p	T2	p	T3	p	T4	p	T5	p	T6	p
xperia	0.078	xbox	0.035	techno	0.056	new	0.028	apple	0.036	world	0.039
tablet	0.065	tesco	0.032	network	0.050	samsung	0.021	iphone	0.031	event	0.016
verizon	0.035	sony	0.032	devices	0.049	tech	0.021	world	0.025	peace	0.015
technology	0.034	features	0.027	surveillance	0.049	launch	0.021	iphone6	0.019	science	0.013
sony	0.033	apple	0.022	wifi	0.020	report	0.015	peace	0.019	best	0.013
wi-fi	0.033	friends	0.020	potential	0.019	sm-a500	0.015	new	0.019	innovation	0.013
google	0.027	mi	0.017	scans	0.018	smartphone	0.015	innovation	0.019	costumers	0.011
major	0.018	october	0.019	cyborg	0.018	old	0.014	review	0.014	future	0.011
hell	0.017	6	0.019	unplug	0.018	series	0.014	samsung	0.014	conference	0.010
network	0.017	new	0.019	wi-fi	0.017	technews	0.014	know	0.013	wrangle	0.010

Table 15: Extracted topics from technology dataset for phrase ranking

The top labels for each topic are:

- $label_{T1}$: xperia sony technology, score=0.111354320918
- $label_{T2}$: new snap friends features, score=score=0.042114685245
- $label_{T3}$: potential surveillance devices, score=0.077669646858
- $label_{T4}$: new samsung galaxy smartphone,score=0.052947947959
- $label_{T5}$: apple new iphone, score=0.05294894779593
- $label_{T6}$: annual event innovation, score=0.0249383150644

As we can see again our zero-order function assigns semantically meaningful labels to our topics and can provide better understanding of a topic without the need to manually label the topics.

5.4 Unfolding communities

In this chapter we are going to analyze the results extracted by our community detection algorithms. We have created a new dataset of 7135 tweets related with economy. Using our Python NetworkX script in order to create the network graph we export a graph of 7135 nodes (users) and 5165 edges (interactions). We can then compute the degree distribution in our large graph and then apply our community detection algorithms using modularity classes to cluster our graph into communities.

Results:

Average degree: 0.724

Average path length: 1.945

Clustering coefficient: 0.048

Number of Communities: 3007

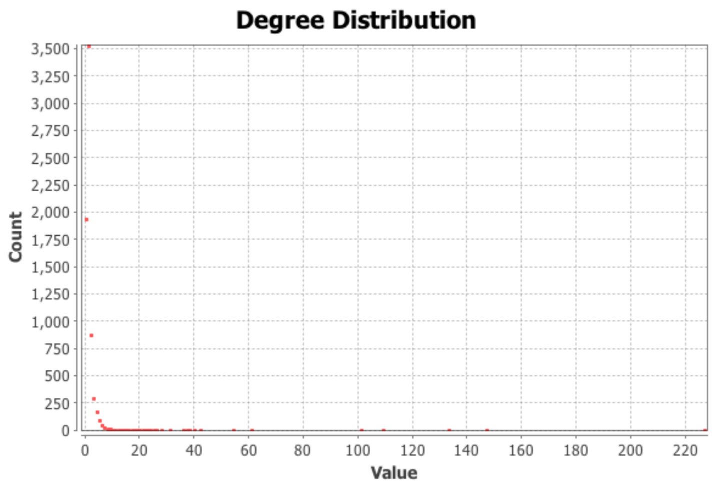


Figure 15: Degree distribution

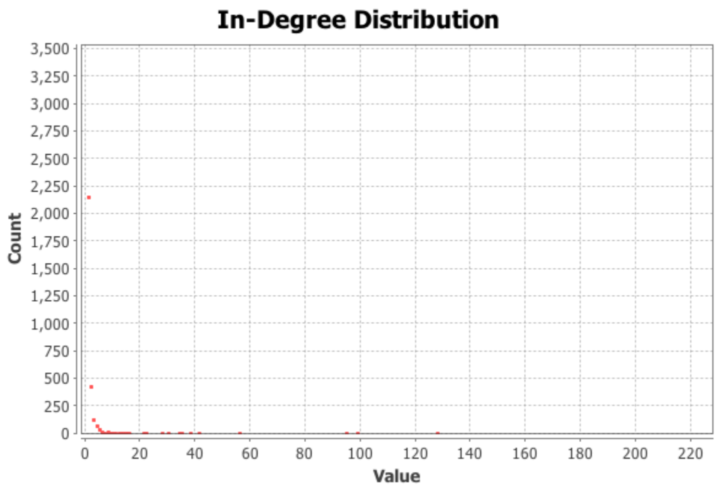


Figure 16: In-degree distribution

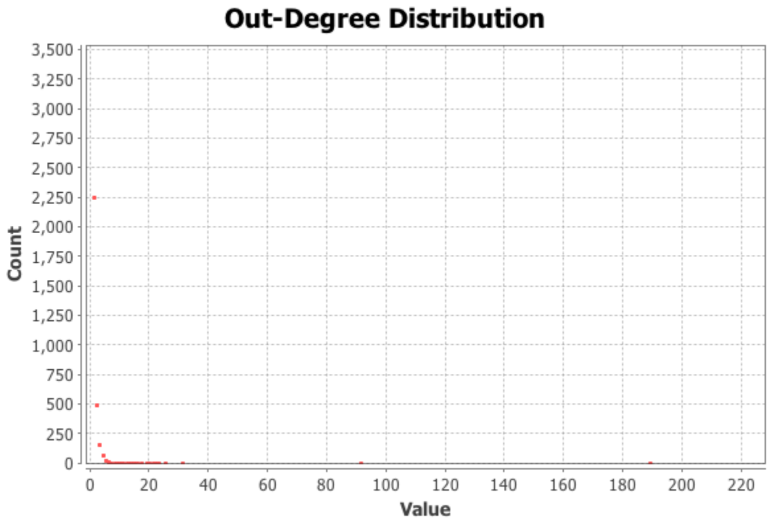


Figure 17: Out-degree distribution

The average degree is 0.724 which means that on average a Twitter user in our network has less than one connection (interaction) with another Twitter user which us relatively low but was expected due to the nature of social networking. Computing the average path length we get 1.945, which is relatively short, meaning that the clusters are well connected. The clustering coefficient is 0.048, which is relatively small and means that our network is not well inter-connected that leads to the conclusion that propagation of information is difficult in this network between the nodes. In order to provide better visualization we will need to apply filtering in our network. The final step is to make the community more visible by calculating modularity measure. Modularity will show the clusters of nodes that are more densely connected together than the rest of the network. Using the modularity module we are going to use 5.0 as a resolution (default is 1.0) in order to acquire better quality in our extracted partitions with bigger communities.

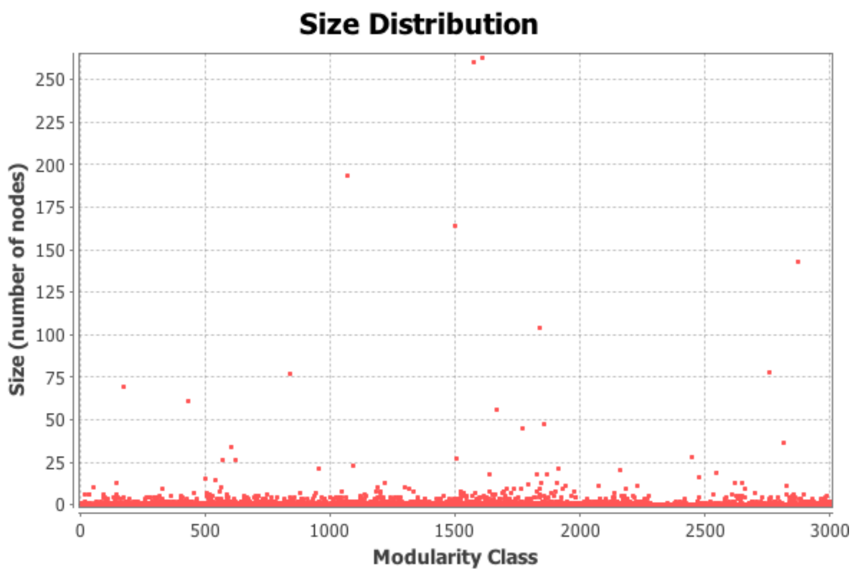


Figure 18: Size distribution

Using our extracted modularity classes we can now display the graph by assigning a different color to each class and we can use our degree distribution in order to assign different sizes to nodes based on their degree. This way as we already proposed in our methodology, the higher degree and larger nodes will indicate the most 'important' nodes in our communities.

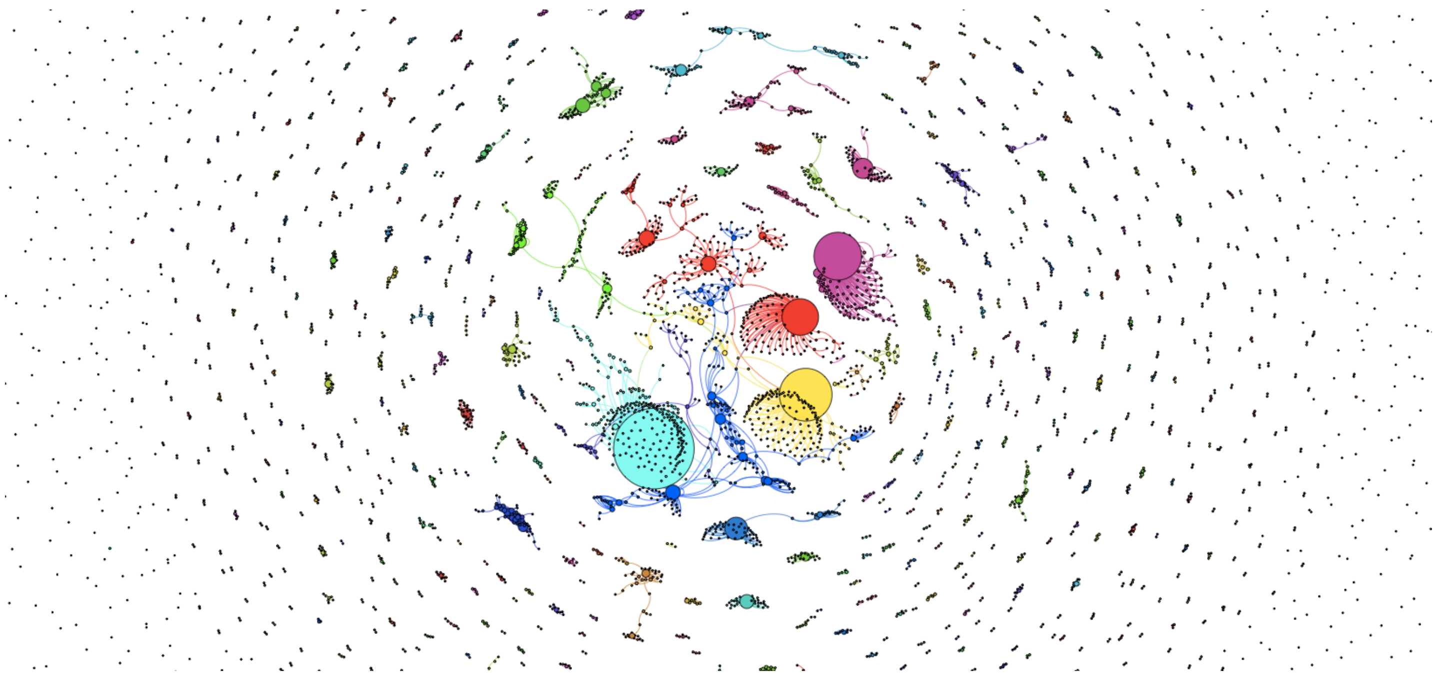


Figure 19: Extracted network (all communities)

As our clustering coefficient, and the number of communities in our network already had indicated, a large percentage of unimportant communities are included in our network. We can see that the most important ones are clustered in the center and the most unimportant ones are in the perimeter of our network. Now we will apply some filtering in order to remove these unimportant networks and only visualize relatively large partitions. The threshold that we are going to apply will be that we will only keep community partitions that are larger than 0.5% of the total population in our networks. Applying this filtering we get a new graph with 1610 nodes and 2124 edges.

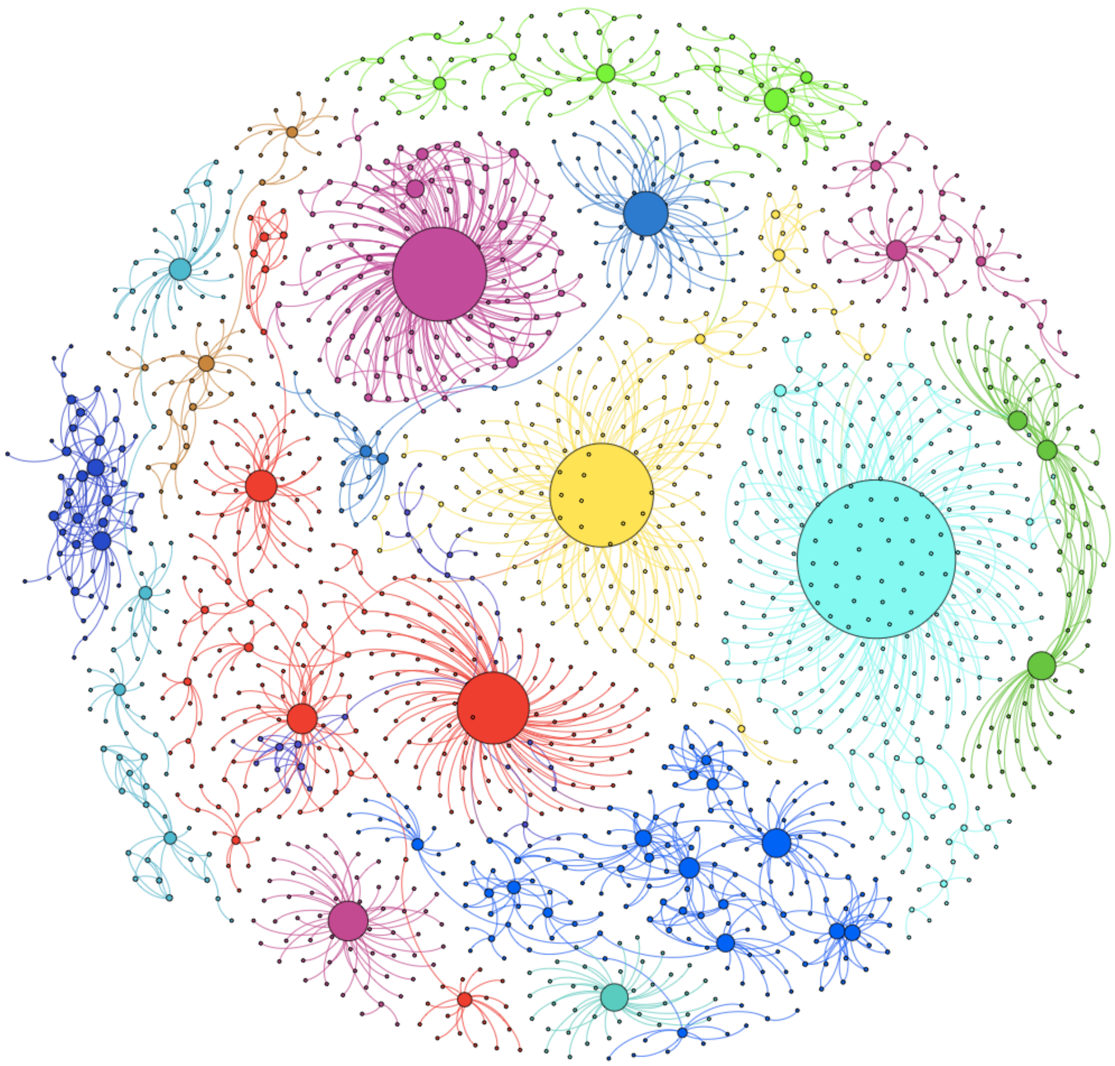


Figure 20: Extracted network (only interesting communities)

As we can see the communities are well ordered and they are relatively large meaning that extracting information form these communities would be useful. As we already proposed in our methodology, we have developed a way of assigning the topics of interest in each community with a percentage of relevance. The extracted topics among with their highest scored labels from our new economy dataset are:

T0	p	T1	p	T2	p	T3	p	T4	p	T5	p
japanese	0.024	world	0.023	asia	0.056	world	0.031	omojuwa	0.031	eurozone	0.037
growth	0.022	omojuwa	0.019	unemploymer	0.050	adani	0.030	gas	0.025	fragile	0.025
recession	0.021	nigerian	0.019	russian	0.049	uk	0.018	fly	0.025	europe	0.024
resigned	0.016	global	0.018	longterm	0.049	billion	0.018	aero	0.024	crisis	0.019
economy	0.015	need	0.017	people	0.020	cup	0.017	abuja	0.019	best	0.019
money	0.013	economy	0.017	forbes	0.019	rugby	0.015	afford	0.018	poverty	0.018
laundering	0.012	gratification	0.017	money	0.018	loan	0.015	moving	0.017	italy	0.017
murders	0.012	leveraging	0.016	fool	0.018	cameron	0.013	price	0.016	future	0.017
ptiofficial	0.012	us	0.016	part	0.018	economy	0.013	advance	0.013	economy	0.016
japan	0.012	business	0.015	economic	0.017	sense	0.013	global	0.013	investment	0.016

Table 16: Extracted topics from our new economy dataset for phrase ranking

The label assignment in each topic is:

- $label_{T0}$: japanese economy, score=0.043996084379
- $label_{T1}$: nigerian economy, score=0.022955712963
- $label_{T2}$: longterm unemployment spread, score=0.081185296665
- $label_{T3}$: rugby world cup, score=0.052455800831
- $label_{T4}$: lower gas price, score=0.018443751034
- $label_{T5}$: eurozone crisis, score=0.063691588198

As we can see out extracted topics are semantically meaningful and represent a category of economy news. Some topics are abstract like topic 4, which is probably related with topic 2 and $label_{T4}$ probably did not capture the semantic meaning of the topic accurately, however all the other automated topic labels are representing their topics correctly. Now we can use our network to visualize these topics and their distribution over our communities:

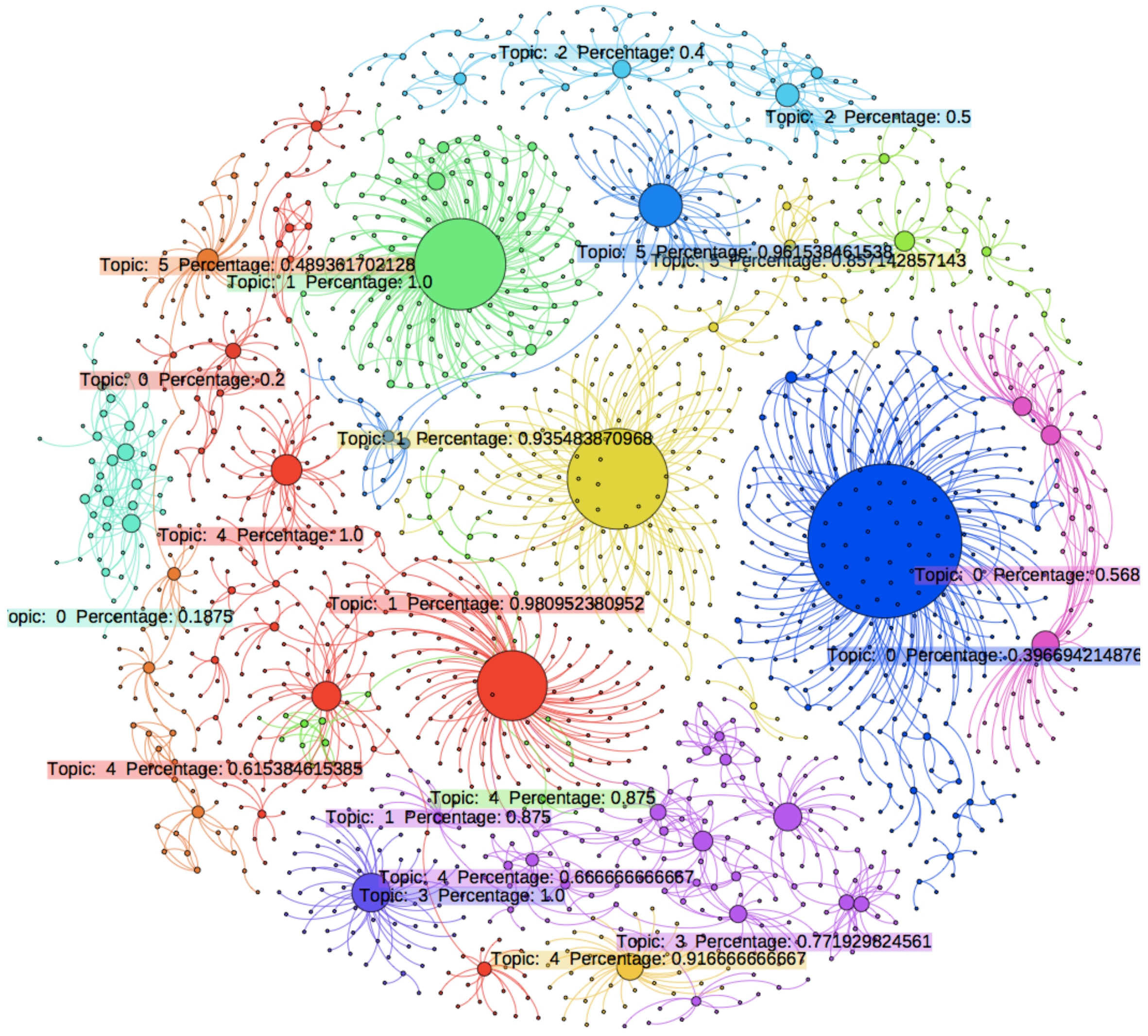

Topic: 2 Percentage: 0.4
Topic: 2 Percentage: 0.5
Topic: 5 Percentage: 0.489361702128
Topic: 1 Percentage: 1.0
Topic: 5 Percentage: 0.961538461538
Topic: 5 Percentage: 0.857142857143
Topic: 0 Percentage: 0.2
Topic: 1 Percentage: 0.935483870968
Topic: 4 Percentage: 1.0
Topic: 0 Percentage: 0.568
opic: 0 Percentage: 0.1875
Topic: 1 Percentage: 0.980952380952
Topic: 0 Percentage: 0.396694214876
Topic: 4 Percentage: 0.615384615385
Topic: 4 Percentage: 0.875
Topic: 1 Percentage: 0.875
Topic: 4 Percentage: 0.666666666667
Topic: 3 Percentage: 1.0
Topic: 3 Percentage: 0.771929824561
Topic: 4 Percentage: 0.916666666667

Figure 21: Final extracted network (only interesting communities with labels)

He have exported our graph using Python's NetworkX library to GEPHI and used OpendOrd module as a layout algorithm to provide enhanced visual results. Now we can see the distribution of the topics among our communities. Regarding the actual users of the communities whose names are not shown for privacy reasons, the nodes are interacting with each other and discussing topics like Japanese economy or Eurozone crisis into their communities. An interesting observation in this graph is that the community with red color is discussing mostly about topic 1 and topic 4 and as we already had observed this topics are relatively similar which indicates the effectiveness of our methods.

6. Conclusions and Future Work

In this thesis we have showed approaches for clustering content from Twitter using probabilistic topic modeling methods such as Latent Dirichlet Allocation and then recorded how these topics are distributed in communities in the Twitter graph. This way we have been able to identify which topics and which users are more influential. Using automatic topic labeling we have enhanced the semantics of our extracted multinomial topics providing a better understanding of our topics. Utilizing natural language processing techniques such us chunking/shallow parsing has helped us to generate grammatically correct phrases from our dataset as candidate labels for our topics. Then we have developed a scoring function in order to assign the most semantically similar labels to our topics. Using community detection algorithms we have been able to identify which are the topics of interest for each community and in what percentage, providing their visualization in the Twitter graph. We have collected various datasets using Twitter's API and used open source tools to reinforce our research.

For the requirements of our research goals and for the process of application development we have used a plethora of tools, modules and libraries. **MongoDB** as a database management system, **Python 2.7.8** as the programming language to implemented our methods and our ideas, **tweepy** as a Python's library to interpret the Twitter API, **scipy** and **numpy**, which are open-source libraries for mathematics, **scikit-learn** and **gensim** which are tools for data mining and machine learning for the implementation of probabilistic topic models such as LDA, regular expressions which is a Python's native module (**regex**) and was used for pattern matching, **NLTK** which is a Python's library that helped us to implement natural language techniques, **NetworkX** which is a Python's software package for the manipulation of complex networks help us to generate our graphs and to implement graph operations, **communityAPI** which is a module for NetworkX helped us to apply community detection algorithms to our graphs, **Gephi** which is an open-source network analysis and visualization software helped us to analyze our graphs and use state of the art layout algorithms and to visualize our graphs.

Due to the limitations of Twitter's API which only allows a small percent of the actual streaming data in Twitter to be accessible for research or third party applications our data collection process has been stalled without being able to access the total amount of Twitter streaming data. Ideally we would collect continuously all the available data for a certain domain of our interest (such as economy or technology using `streamListener()` in our Twitter's API) and we would remove the unnecessary data from our database in parallel with data collection until a certain amount of data is reached. Another approached would be to use an already "trimmed" large dataset however as the amount of tweets are rapidly increasing day by day these datasets are becoming valuable assets for companies. Despite these limitations we have been able to develop a fully working prototype that can extract valuable information from Twitter data. An application like this could be a valuable asset to companies that want to perform costumer analysis or to identify the interests of groups and communities in order to perform market research or to analyze sociological events like political parties formation. Using our *zero-order relevance scoring function* we have been able to assign semantically meaningful labels to our topics however for future work we could leverage our *zero-order* to a *first-order relevance scoring function* using the Kullback Leibler (KL) divergence to evaluate the similarity of a generated label to a topic. Moreover for future work we can extend our research and include sentiment analysis and named entity recognition methods to provide another semantic layer in our extracted network that can identify name entities such as brand names, geo-locations, people and display the these entities in our network providing a semantic analysis from the nodes (users) in the community that this entity is found. Again this could be a valuable asset to companies and businesses that want to discover how costumers or certain communities are reacting to brand names or events. Previous researchers have used sentiment analysis for suicide prevention and gender recognition, which indicates the variety and depth that these fields of study can reach not only for the benefit of companies but also for the improvement of society.

Bibliography

[1] Hyun-Woo Kim. (2010). Data Mining on Microblogged Information: Gender Recognition and Suicide Prevention. Master Thesis. Pennsylvania State University

[2] Qiaozhu Mei, Xuehua Shen, Chengxiang Zhai. (2007). Automatic Labeling of Multinomial Topic Models. University of Illinois. pages [2-4]

[3] Matthey Michelson, Sofus A. Macskassy (2010). . Discovering Users' Topics of Interest of Twitter: A First Look. Fetch Technologies

[4] Michael Mathioudakis, Nick Koudas. (2010). TwitterMonitor: Trend Detection over the Twitter Stream. University of Toronto.

[5] Ivan Marcin, Sam Shiu. (2012). Extracting Topic Trends and Connections: Semantic Analysis and Topic Linking in Twitter and Wikipedia Datasets. Stanford University.

[6] Reza Zafarani, Mohammad Ali Abbasi, Huan Liu. (2014). Social Media Mining. Arizona State University.

[7] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre. (2008). Fast Unfolding of Communities in Large Networks. Univeristy Pierre et Marie Curie. pages 2-7

[8] R. Lambiotte, J. C. Delvenne, M. Barahona. (2009). Laplacian Dynamics and Multiscale Modular Structure in Networks

[9] Santo Fortunato. (2010). Community Detection in Graphs. Complex Networks and Systems Lagrange Laboratory, ISI Foundation. pages [4-7]

[10] Matthew A. Russel. (2013). Mining the Social Web. O'Reilly Media. pages [183-213],[370-380]

[11] Matthew A. Russel. (2011). 21 Recipes for Mining Twitter. O'Reilly Media. pages [7-15]

[12] Steven Bird, Ewan Klein, Edward Loper. (2009). Natural Language Processing with Python. O'Reilly Media. pages [79-106], [273-285]

[13] Bretonnel K. Cohen, Lawrence Hunter. (2008). Getting Started in Text Mining. PLoS Comput Biol 4(1):e20. doi:10.1371/journal.pcbi.0040020

[14] Hastie Trevor, Tibshirani Robwer, Friedman Jerome. (2009). The Elements of Statistical Learning: Data Mining, Inference and Prediction.

[15] David M. Blei, John D. Lafferty. (2009). Topic Models. Princeton and Carnegie Mellon University. pages [2-4]

[16] David M. Blei (2010). Introduction to Probabilistic Topic Models. Princeton University. pages [6-8]

[17] Reza Zafarani, Mohammad Ali Abbasi, Huan Liu (2014). Social Media Mining. Cambridge University Press. chapter 1: [1-5] , chapter 2: [29-36], chapter 3: [74-77]

[18] Di Battista,Peter Eades, Roberto Tamassia, Ioannis G. Tollis. (1999). Graph Drawing: Algorithms for the visualization of Graphs. Prentice Hall, ISBN 978-0-13-301615-4

[19] Burt R. S. (2005). Brokerage and closure: An introduction to social capital. Oxford University Press.

[20] Ghosh R.,Lerman K. (2011). Parameterized centrality metric for network analysis. Physical Review E. 83(6), 0066118

[21] Newmann M. (2011). Networks: An introduction. Oxford University Press.

[22] Mariam Adedoyin-Olowe, Mohamed Medhat Gaber, Frederic Stahl. (2014). A Survey of Data Mining Techniques for Social Network Analysis. Robert Gordon University.

[23] Terry Winograd. (1971). Procedures as a representation for Data in a Computer Program for Understanding Natural Language. MIT AI Technical Report 235.

[24] S. Martin, W. M. Brown, R. Klavans, and K. Boyack. (2011). OpenOrd: An Open-Source Toolbox for Large Graph Layout. SPIE Conference on Visualization and Data Analysis (VDA).

Appendix

Appendix A' Python Scripts:

The most important Python scripts are going to be presented in this section.

A.1 File: connectTweepyMongo.py

```
import tweepy
import sys
import pymongo

consumer_key="*****"
consumer_secret="*****"

access_token="******"
access_token_secret="*****"

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)

class CustomStreamListener(tweepy.StreamListener):
    def __init__(self, api):
        self.api = api
        super(tweepy.StreamListener, self).__init__()

        self.db = pymongo.MongoClient().economyLAST

    def on_status(self, status):
        print status.text

        data ={}
        data['user']=status.user.screen_name
        data['text'] = status.text
        data['created_at'] = status.created_at
        data['geo'] = status.geo
        data['source'] = status.source
        data['entities']=status.entities

        self.db.Tweets.insert(data)
    def on_error(self, status_code):
        print >> sys.stderr, 'Encountered error with status code:', status_code
        return True # Don't kill the stream

    def on_timeout(self):
        print >> sys.stderr, 'Timeout...'
        return True # Don't kill the stream

sapi = tweepy.streaming.Stream(auth, CustomStreamListener(api))
sapi.filter(track=['economy'])
```

A.2 connecttweepycsv.py:

```
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
import time

ckey = ***
csecret = ***
atoken = ***
asecret = ***

class listener(StreamListener):

        def on_data(self, data):
                try:
                        #print data


                        #trimmed to getr only the tweet
                        tweet = data.split(',"text":"')[1].split('","source')[0]
                        print tweet

                        # saveThis=str(time.time())+'::'+tweet
                        #testing without time at first
                        saveThis=tweet
                        saveFile=open('economyLAST.csv','a')
                        saveFile.write(saveThis)
                        saveFile.write('\n')
                        saveFile.close()
                        return True

                #if internet drops for example
        except BaseException, e:
            print 'failed ondata,',str(e)
            #wait for reconnection
            time.sleep(1)



    def on_error(self, status):
        print status

auth = OAuthHandler(ckey, csecret)
auth.set_access_token(atoken, asecret)
twitterStream = Stream(auth,listener())
twitterStream.filter(track=['economy'])
```

A.3 chunkerPhraseGeneration.py

```
import nltk
import re
import time
import csv


#Regular Expressions:
#? = 0 or 1 rep
#* = 0 or more rep
#+ = 1 or more rep


tweetList = []
Phrase = []
phrases = []
phr=''
urlone='http:'
urltwo='\\'

output = open('Stopwords_for_Tweets.txt', 'r')
#read from saved tweets CSV
with open('economyLAST.csv','rU') as csvfile:
    tweetreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
    for row in tweetreader:

        #Removing all links-URLs from Tweets
        for word in row:
            if urlone in word or urltwo in word:
```

```
                        row.remove(word)

                x=' '.join(row)
                tokens = nltk.word_tokenize(x)

#               # print (tokens)

                y=' '.join(tokens)

                tweetList.append(y)

#               # time.sleep(0.1)


stoplist=set(output.read().split())
texts = [[word for word in tweet.lower().split() if word not in stoplist]
        for tweet in tweetList]


all_tokens = sum(texts, [])
tokens_once = set(word for word in set(all_tokens) if all_tokens.count(word) == 1)
texts = [[word for word in text if word not in tokens_once]
        for text in texts]


for tweet in texts:

                        # tokenized = nltk.word_tokenize(tweet)


                        tagged = nltk.pos_tag(tweet)


                        #find what adjective modifies a proper noun

                        grammar = r"""Phrase:
{<NN\w?>*<DT\w?|NN\w?>*<JJ\w?|VG\w?>+<NN\w?>+<JJ\w?|VG\w?>*}"""
                        # grammar = r"""Phrase:
{<JJ\w?|VG\w?|NN\w?|DT\w?>+<NNP>+}"""
                        # grammar = r"""Phrase: {<RB.?>*<VB.?>+<NNP>}"""


                        chunkParser = nltk.RegexpParser(grammar)
                        chunked = chunkParser.parse(tagged)

                        print '-------------------'
```

```
            for i in chunked.subtrees(filter=lambda x: x.node=='Phrase'):
                a=i.leaves()
                # print a
                for j in a:
                    # print j
                    for k in j:
                        print k

                        phrases.append(k)
                        break
                phrases.append(',')

                print '********************'


for ph in phrases:
    print ph
    text=open('phraseseconomyLast.txt','a')
    text.write(ph + ' ')
    text.close
    # chunked.draw()

textz=open('phraseseconomyLast.txt','r')
phrzs=textz.read().split(',')

for p in phrzs:
    print p

    saveFile=open('phraseseconomyLast.csv','a')
    saveFile.write(p)
    saveFile.write('\n')
    saveFile.close()
```

A.4 topicExtractionLDAphraseRanking.py

```
from gensim import corpora, models, similarities
from gensim.models import hdpmodel, ldamodel
from itertools import izip
import csv
import re
import nltk
import math
from operator import itemgetter
import unicodedata
import time
```

```
tweetList = []
urlone='http:'
urltwo='\\'
topics=[]
score=0.0
s= 0.0
m=0
scoretable=[[],[],[],[],[],[],[],[],[],[],[]]
phrzs=[]
finalist=[]

#read from saved tweets CSV
with open('economyLAST.csv','rU') as csvfile:
        tweetreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
        for row in tweetreader:


                #Removing all links-URLs from Tweets (3 times)
                for word in row:
                        if urlone in word or urltwo in word:
                                row.remove(word)
                                # print ("----------Word REMOVED------------")
                                # print word

                x=' '.join(row)
                # print (x)
                tokens = nltk.word_tokenize(x)
                # print (tokens)

                y=' '.join(tokens)

                #creating the tweet list
                tweetList.append(y)
                # print(tweetList)

# remove common words and tokenize
output = open('Stopwords_for_Tweets.txt', 'r')
# stopwordz=output.read().split(' ')

stopwordz=output
stoplist = output
# # stoplist = set('for a of the and to in technology - '.split())
stoplist=set(output.read().split())
texts = [[word for word in tweet.lower().split() if word not in stoplist]
      for tweet in tweetList]
```

```
# # remove words that appear only once
all_tokens = sum(texts, [])
tokens_once = set(word for word in set(all_tokens) if all_tokens.count(word) == 1)
texts = [[word for word in text if word not in tokens_once]
        for text in texts]

dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]

lda = ldamodel.LdaModel(corpus, id2word=dictionary, num_topics=7)
corpus_lda = lda[corpus]

# for l,t in izip(corpus_lda,corpus):
#   print l,"::",t
#print topics
for i in range(0, lda.num_topics):


          print lda.print_topic(i)

          topics.append(lda.print_topic(i))

time.sleep(7)
for t in topics:
          # print ph
          txt=open('economyLASTtopics.txt','a')
          txt.write(t + '\n')
          txt.close

for topic in topics:

                    print 'New Topic:-------------------------------'

                    with open('phraseseconomyLast.csv','rU') as csvfile:

                              phrases = csv.reader(csvfile, delimiter=' ', quotechar='|')
                              for phrase in phrases:
                                        joinphrase=' '.join(phrase)

                                        # print 'Phrase',phrase,'for Topic',topic,'SCORE:',score
                                        # score=0.0
                                        b=topic.split('+')

                                        # phrzs.append(joinphrase)
```

```
                for item in b:

                        a=item.split('*')
                        for p in joinphrase.split():

                                print p,a[1]
                                # time.sleep(0.02)
                                if str(p) in str(a[1]):
                                    print 'Probability:',a[0]
                                    print 'Phrase:', phrase
                                    print 'Topic:' ,topic

    score=float(math.log10(float(a[0])+float(1)))+score

                                    # time.sleep(2)

                print 'Phrase',phrase,'for Topic',topic,'SCORE:',score

                scoretable[m].append(score)

                score=0.0

                # time.sleep(1)
        m=m+1

with open('phraseseconomyLast.csv','rU') as csvfile:

            phrases = csv.reader(csvfile, delimiter=' ', quotechar='|')
            for phrase in phrases:
                joinphrase=' '.join(phrase)
                phrzs.append(joinphrase)


print phrzs
print scoretable

#range = num of topics
for i in range(len(num_topics)):
    try:

        max(scoretable[i])

        print max(scoretable[i]),scoretable[i].index(max(scoretable[i]))
        print phrzs[scoretable[i].index(max(scoretable[i]))]
        finalist.append(phrzs[scoretable[i].index(max(scoretable[i]))])
        print lda.print_topic(i)
        except:
                pass
```

A.5 graphgenerationwithlabels.py

```
from __future__ import division
from pymongo import Connection
import pymongo
from pymongo import MongoClient
import time
import json
import networkx as nx
import matplotlib.pyplot as plt
import community
import random
import csv
import nltk
import math


scoretable=[[],[],[],[],[],[],[],[],[],[],[]]
communitytopicscores=[]
st=[0]
score=0.0
a={}
d=['']
i=0
tweet={}
stopList = open('Stopwords_for_Tweets.txt', 'r').read().split()
tweetList=[]
m=0
i=0
perc=0.0
topic1=0
topic2=0
topic3=0
topic4=0
topicstats=[]
topiccounts=[]
nodesdegreelist=[]
nodesnameslist=[]

textz=open('economyLASTtopics.txt','r')
topcs=textz.read().split('\n')
```

```
c=Connection()
db=c.economyLAST  #my database #Another DB: EbolaTest,newEbolaDB,economyLAST
tweets=db.Tweets

#creating the empty graph for networkX
mynetwork=nx.Graph()
mentions=db.Tweets.entities.user_mentions
a=tweets.find()
test=tweets.entities.user_mentions.find()

for t in a:

        mynetwork.add_node(t['user'],tweet=t['text'])

        if len(t['entities']['user_mentions'])!=0:
                try:

                  for i in range (0, 10):

                          mynetwork.add_edge(t['user'],
t['entities']['user_mentions'][i]['screen_name'])

                except:
                        pass

#first compute the best partition
partition = community.best_partition(mynetwork)
#drawing
size = float(len(set(partition.values())))
# pos = nx.spring_layout(mynetwork)
count = 0
# print partition.values()
for com in set(partition.values()) :
    count = count + 1
    list_nodes = [nodes for nodes in partition.keys()
                                if partition[nodes] == com]


    #random colors in every loop
    t='#'+str(hex(random.randint(0,16777216))[2:])


    # print len(list_nodes)
    if (len(list_nodes)>5):
            for node in list_nodes:
                    nodesdegreelist.append(mynetwork.degree(node))
                    nodesnameslist.append(node)

                    try:
p
                                        d=mynetwork.node[node]['tweet']

                                        tokens=nltk.word_tokenize(d)
                                        filteredtext = [t for t in tokens if t.lower() not in stopList]
                                        tokenizedtweet=' '.join(filteredtext)
                                        print 'TWEET',tokenizedtweet


                                        st=[]

                                        for topic in topcs:

                                                  for word in tokenizedtweet.split():

                                                                      b=topic.split('+')
                                                                      for item in b:

                                                                                a=item.split('*')

                                                                                if word in a[1]:



                                                                                          print
                                                                                          print 'Probability:',a[0]
                                                                                          print 'Tweet:',
tokenizedtweet.split()
                                                                                          print 'Topic:' ,topic
                                                                                          print

          score=float(math.log10(float(a[0])+float(1)))+score

                                                  st.append(score)
                                                  score=0.0

                    except:
                              pass


                    if(max(st)>0):
```

```
                                    print 'topic number:',st.index(max(st)),'for tweet in
community'
                                    print max(st)
                                    print st.index(max(st))
                                    communitytopicscores.append(st.index(max(st)))

            print 'Community Topic Scoring Results:'
            for topic in communitytopicscores:

            if (len(communitytopicscores)>2):

                     for i in range (0, len(num_topics))
                        topiccounts.append(communitytopicscores.count(i))
                        i=i+1

                     perc=max(topiccounts)/len(communitytopicscores)

                     print 'MAX: ',max(topiccounts),'TOPIC:
',topiccounts.index(max(topiccounts)), 'PERCENTAGE: ',perc


                     print 'NODES DEGREES'
                     for nodedegree in nodesdegreelist:
                           print nodedegree

                     maxdegree=max(nodesdegreelist)
                     # print maxdegree
                     print 'POSITION'
                     print nodesdegreelist.index(max(nodesdegreelist))

                     print 'NODES NAMES'
                     for nodename in nodesnameslist:
                           print nodename

                     print 'HIGHEST NODE DEGREE NAME (MOST INFUENCIAL IN
COMMUNITY):'
                     print nodesnameslist[nodesdegreelist.index(max(nodesdegreelist))]
                     print 'MOST RELEVANT TOPIC ON HIS COMMUNITY:'
                     print topiccounts.index(max(topiccounts)), 'WITH
PERCENTAGE:',perc


                     topicassignment='Topic:  '+str(topiccounts.index(max(topiccounts)))+'
Percentage: '+str(perc)
                     print topicassignment
                     mynetwork.add_node(node,topic=topicassignment)
            communitytopicscores=[]
            topiccounts=[]
            nodesdegreelist=[]
            nodesnameslist=[]
            print 'NEW COMMUNITY'

nx.write_gml(mynetwork,"NEWresults.gml")
```

Appendix B’ Figures and Tables: