Universidad Nacional de Educación a Distancia

Escuela Técnica Superior de Ingeniería Informática

Master Thesis

New clustering proposals for Twitter based on the Louvain algorithm

Author:
Jordi Colomer Matutano

Supervisor:
Raquel Martínez Unanue

A thesis submitted in fulfilment of the requirements for the degree of Master in Artificial Intelligence and Computer Systems

in the program

Máster Universitario En I.A. Avanzada: Fundamentos, Métodos y Aplicaciones 2013/2014

September 7, 2014

Contents

List of Figures

List of Tables

Abstract

Document clustering is a problem that has been successfully solved in contexts where the size of the documents is big. When the size of the documents has a limit of 140 characters as in Twitter, then this problem becomes more challenging. In this work we will address this problem, which is one component of a larger task of reputation monitoring of an entity using real time data from Twitter.

We use the idea of extended modularity for community detection that allows us to choose the granularity of the communities found. We introduce a new Fuzzy Jaccard similarity measure that calculates the similarity at the semantic level of the tweets being compared. Our proposed algorithms based on modularity maximization consistently outperform other proposed solutions in 2013 edition of the Replab competition reputation monitoring using Twitter.

Moreover, we will give a constructive critique to the B-Cubed F-Measure evaluation metric and prove that it fails to comply with a natural constraint that we call the double improvement consistency. We will show a different flaw that we will exploit with a simple and apparently naive algorithm that will rank number one by the B-Cubed F-Measure in the Replab 2013 competition.

Acknowledgments

I would like to express my gratitude to my supervisor Raquel Martínez Unanue for the useful comments, remarks and engagement through the learning process of this master thesis.

Chapter 1

Introduction

In this chapter, we will describe the problem, give the motivation and define the objectives of this master thesis.

1.1 Problem description

Clustering is the task of grouping a set of observations into subsets with elements that are similar among them. Unlike the task of document classification, the subsets or classes are not predefined. It is a well understood problem and considered solved when we are clustering large documents such as articles, webpages or books, but becomes a more challenging problem when the size of the documents is very short, as in Twitter. An additional difficulty that we find when we analyze documents in Twitter is the amount of noise that we can find in the messages, as some of the messages carry little or no valuable information. Moreover, messages are written in different languages, which requires some kind of translation in order to detect similar messages when they are not written in the same language. Due to the 140 character limitation, often word abbreviations are used, and in other cases words are misspelled. Some words have a special meaning in Twitter: hashtags (words that start with the character #), user references (words starting with the character @), and emoticons. These cases might require a special treatment. Given the short nature of the messages, metadata can prove to be a valuable resource of information for the clustering task such as temporal data, the author or the author network of followers.

1.2 Motivation

In the era of communication, Internet, digital information and big data, data mining and text mining are becoming increasingly relevant and powerful. Document clustering of digital texts is a difficult task but holds many possibilities that we only began exploring. One of them is to monitor the reputation of a public entity such as a company, celebrity, or organization by analyzing the

stream of text data that Twitter offers in real time. In a world where the powerful consumers and the general public are constantly evaluating the actions of such entities, maintaining and keeping track of their reputation becomes a crucial task that allows public entities to react appropriately to maintain a positive public image. Additionally, to maintain a reputation, it is important to be able to respond quickly to some events. Traditional reputation analysis has been mostly a manual task, but digital data and computers can enable us to process large amounts of information in real time.

Topic detection is the task of categorizing a message under a category. This task can be accomplished by using clustering techniques.

The full monitoring task is composed of four subtasks: (a) The filtering subtask that consists in determining which tweets are related to the entity and which ones are not. (b) The polarity subtask, where the goal is to decide whether the tweet content has positive, negative or neutral implications for the entity's reputation. (c) The topic detection subtask that clusters related tweets into groups by topic. And (d) the priority assignment subtask that detects the relative priority of topics. In this work, we will focus on the document clustering subtask for Twitter.

The task of clustering has been solved in other contexts, but it is still an open problem in the context of tweets. This work's aim is to shed more light on this unsolved problem.

In order to successfully evaluate a reputation management system or one of its components, we need a reliable method to measure its performance. Several metrics have been proposed, B-Cubed F-Measure being the most generally accepted at the time of this writing. Although it has been proven to fullfill a number of desirable constraints [29], it lacks scientific community consensus to be useful for the tweet clustering task. In this work, we will provide more evidence for the suitability of the B-cubed F-Measure for the particular task of tweet clustering.

1.3 Objectives

Complex network analysis has been successfully used in many areas of research. In particular, community detection algorithms based on Modularity maximization have been used in many real networks such as social, metabolic or citation networks. Approaches based on hierarchical clustering or the Latent Dirichlet allocation have been proven useful with different degrees of success for the task of tweet clustering. Our hypothesis is that community detection based on Modularity maximization can also be useful for this task. Below, we present a list of objectives for this study:

- The first objective is to review the literature about topic detection in short texts to detect possible new lines of research and propose a new approach.
- In order to test our hypothesis, we will design and implement a system capable of clustering tweets based on the Louvain method for community

detection.

- To create a network of tweets, we need to use similarity measures across tweets (conceptualized as nodes of the network) that will calculate the weight of the edges that connect them. We will study existing similarity measures and propose a new one.
- We will study the evaluation metrics used in document clustering to verify their suitability given the characteristics of the problem. We will also review the feasibility of the currently used evaluation metrics for the tweet clustering task.
- We will test lexic as well as semantic (Fuzzy Weighted Jaccard) methods for tweet similarity.
- We will test tweet text expansion (HTML expansion) to avoid the sparsity problem of short texts.
- We will compare the performance of our proposals with other state of the art systems proposed in the Replab 2013 competition, using the Replab 2013 evaluation framework.

These objectives provide a foundation for improving current approaches to the tweet clustering problem. Demonstrating whether the B-Cubed F-Measure has flaws will illuminate future research goals.

Chapter 2

Background

There are many methods or algorithms for the clustering task, such as hierarchical clustering, that connect objects to form clusters using a bottom-up (agglomerative) or a top-down approach (divisive). Centroid-based clustering algorithms, such as the well known k-means clustering, where clusters are represented by a central vector, are also used. Other, less popular, families of algorithms include the distribution-based or the density-based clustering algorithms. In our work, we are going to explore a new kind of clustering algorithm that is based on the concept of complex networks.

This chapter provides the background concepts related to this study. We begin with complex networks as they are central to understand our methodology. We will continue with a description of similarity measures. We will follow with a brief description of weighting schemes. A description of evaluation metrics concludes this section.

2.1 Complex networks

A complex network is a graph with non-trivial topological features such as heavy tail in the degree distribution, high clustering coefficient, assortativity, community structure and hierarchical structure.

A community in a network is a group of nodes with many edges that connect nodes from the same community and few edges that connect nodes from the community to other communities.

There exist many algorithms that automatically detect such communities. Some of them are based on the maximization of the modularity heuristic.

2.1.1 Modularity

Given a network, a partition is a grouping of the nodes. Modularity measures how well a partition separates communities in a network [34]. It is calculated as the proportion of edges that connect nodes from the same community minus

the expected proportion if the edges were randomly distributed, and its range is [-1/2,1). The modularity Q is defined as:

$$Q = \sum_{vw} \left[\frac{A_{vw}}{2m} - \frac{k_v * k_w}{(2m)(2m)} \right] \delta(c_v, c_w) \tag{2.1}$$

where A represents the adjacency matrix of the network (A_{vw} equals 1 when the nodes v and w are connected, and 0 otherwise). k_v and k_w are the degrees of the nodes v and w, respectively. The degree of a node is the number of edges connecting that node, and m is the number of edges. c_v represents the community of node v. $\delta(i, j)$ is 1 if $i = j$ and 0 otherwise.

With the definition of the modularity, we can convert the community detection problem into an optimization problem that can be solved by using any existing general purpose optimization algorithm such as Hill Climbing, Simulated annealing, etc. There also exist many algorithms specially designed to optimize the modularity. We will very briefly describe some notable algorithms.

1. The Girvan Newman algorithm [35] iteratively removes the edge with higher betweenness and considers the connected components to be the partitions. The process stops when the modularity can not be improved.

2. The fast greedy modularity optimization algorithm [36] is a hierarchical agglomeration algorithm that starts with a community per node and iteratively joins the communities with the highest modularity increment.

3. The Louvain method [32] is a greedy optimization method that first optimizes modularity in a local level, and then aggregates the nodes belonging to the same community so that a new network where the nodes are the communities is created. The two steps are repeated iteratively until the modularity is optimized. This method will be described in detail in the next section.

2.1.2 The Louvain method

The Louvain method [32] is an algorithm that is able to find high modularity partitions in large networks in a short time. It works by iteratively running two phases: the first phase tries to find a partition of the network that maximizes the modularity, and the second phase unfolds each partition found into a node and connects these new nodes with weights representing the connectivity of the partitions. The iteration stops when it is not possible to improve the modularity. This methodology gives us access to different resolutions of community detection (each iteration would correspond to a new resolution level). Furthermore, it is possible to efficiently find high modularity partitions for bigger networks than previously possible. Contrary to all the other community detection algorithms, the network size limit when using the Louvain method is limited by storage capacity rather than by computation time.

Assume that we start with a weighted network of N nodes. First, we assign each node to a different community that results in a partition with N communities. Then, for each node i, we evaluate the gain of modularity that would result from replacing the community of i by the community of one of the neighbors of

i. The community that yields the maximum gain of modularity is the chosen one, and i is moved to that community (in case of a tie, a breaking rule is used), but we only change the community of i if the modularity gain is positive, otherwise no changes are applied. This process is applied sequentially for all the nodes in the network, and repeatedly with several passes, until no improvement for modularity can be found. In that case, the phase is complete and we move to the second phase. Several studies show that the ordering of the visiting of the nodes can influence the computation time [32].

Part of the algorithm efficiency is the result of the fact that the increase of modularity obtained by moving a node i from one community B to another community C can be easily computed using the following equation:

$$\Delta Q = [\frac{\sum_{in} + k_{i,in}}{2m} - (\frac{\sum_{tot} + k_i}{2m})^2] - [\frac{\sum_{in}}{2m} - (\frac{\sum_{tot}}{2m})^2 - (\frac{k_i}{2m})^2] \qquad (2.2)$$

where $\sum_{in}$ is the sum of the weights of the links inside C, $\sum_{tot}$ is the sum of the weights of the links incident to nodes in C, k_i is the sum of the weights of the links incident to node i, $k_{i,in}$ is the sum of the weights of the links from i to nodes in C, and m is the sum of the weights of all the links in the network. A similar expression is used to calculate the increase of modularity by removing node i from community B.

The second phase of the algorithm consists in generating a new network where each node represents a community found in the original network by the first phase of the algorithm. The nodes of the new network are connected with links having weights that are the number of links connecting the communities that the nodes are representing in the original network. Within community links in the original network are represented as a self link in the new network with weight equal to the number of within community links.

A pass is the execution of the first and second phases. The passes are iterated until no gain for modularity is possible. This methodology naturally incorporates a notion of hierarchy as communities of communities are built during the process. Figure 2.1 illustrates this idea.

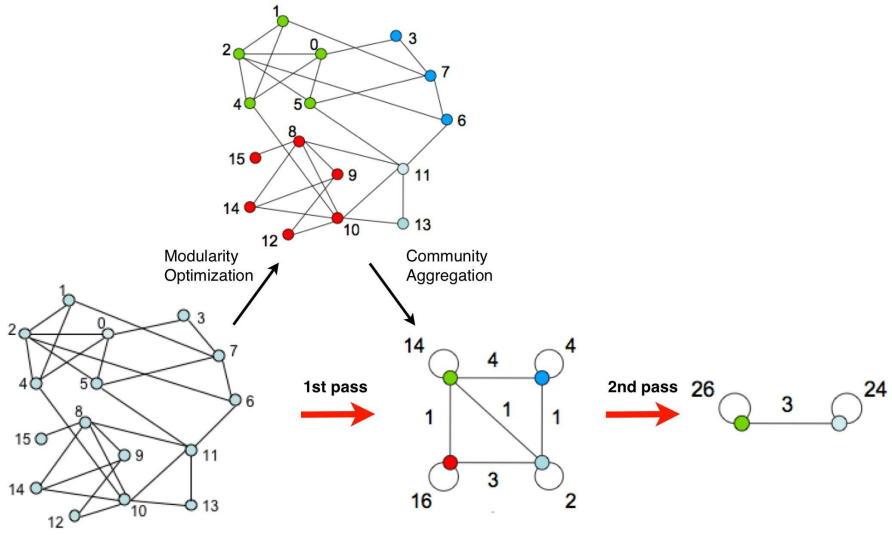


Figure 2.1: Visualization of the steps of the Louvain algorithm. Source: [32]

2.1.3 Resolution limit and multiresolution methods

Modularity optimization may fail to identify unambiguously defined modules smaller than a scale that depends on the number of links of the network and on the degree of interconnectedness of the modules [37]. This is known as the resolution limit. Two approaches have been proposed with the goal of solving the resolution limit problem. One such approach is the addition of a resistance r to every node as a self loop that increases or decreases the tendency of each node to form communities [38]. Another approach entails the addition of a factor γ that multiplies the null-case term in the modularity equation that regulates the relative importance between internal links of the communities and the null model [39].

2.2 Similarity measures

This section describes the Jaccard Index and its variants, the cosine similarity and the Euclidean distance. We have selected the cosine similarity because it is a standard in the context of clustering, the euclidean distance for its simplicity of interpretation, and the Jaccard Index because besides the fact that it has been successfully used in previous works, it will serve as the base for a novel measure that we will call Fuzzy Weighted Jaccard. The latter will allow us to calculate semantic similarities between tweets.

2.2.1 Jaccard index

The Jaccard index or Jaccard similarity coefficient is a similarity measure used for comparing two sets. It is defined as the size of the intersection divided by the size of the union.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.3}$$

2.2.2 Weighted Jaccard index

The Weighted Jaccard index [74] is a generalization of the Jaccard index that uses weights for the terms. This index takes into consideration the weight that gives a measure of the importance of the element. The weighted Jaccard index is calculated as follows:

$$J(A,B) = \frac{\sum_{i \in A \cap B} w_i}{\sum_{i \in A \cup B} w_i} \tag{2.4}$$

where w_i is the weight of the term i.

If we set all weights to the unity, then the Weighted Jaccard index is equivalent to the classical version of the Jaccard index.

At least two alternative definitions of the Weighted Jaccard index are presented in [75] and [76] [77].

2.2.3 Fuzzy Jaccard

The Fuzzy Jaccard [78] is a combination of a term-based and character-based string similarity measure that is a generalization of the Jaccard Index. Unlike the Jaccard Index, terms do not need to be identical to positively contribute to the final score. The computation utilizes a similarity measure across terms.

The calculation is as follows. Given two sets of terms T_1 and T_2, we generate a weighted bi-graph $G = ((T_1, T_2), E)$ where E is a set of weighted edges that connect a vertex from T_1 to a vertex in T_2. The weight of the edges is the edit distances between the terms that edges connect. Edit distance is the minimum number of single-character edit operations (i.e. insertion, deletion, substitution) to transform one term into another. We only keep edges with weights larger than a given threshold δ. We select E so that any two edges have no common vertex and so that the sum of the weights is maximal. The computation is as follows:

$$\text{FJACCARD}_\delta(s_1, s_2) = \frac{|T_1 \widetilde{\cap}_\delta T_2|}{|T_1| + |T_2| - |T_1 \widetilde{\cap}_\delta T_2|} \tag{2.5}$$

where $|T_1 \widetilde{\cap}_\delta T_2|$ is the fuzzy overlap calculated as the sum of the weights in E, and $|T_1|$ and $|T_2|$ are the number of terms in each set.

2.2.4 Cosine similarity

The vector space model is a way of representing text documents as vectors. Each dimension corresponds to a different term existing in the corpus. The value of each dimension of a vector V representing the text T is the weight of the term in T that corresponds to that dimension.

The cosine similarity of two documents is simply the cosine of the vector representation of each document.

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \tag{2.6}$$

The range of the cosine function is $[-1, 1]$, but if the weights of the terms are always positive or 0, then the range is $[0, 1]$.

2.2.5 Euclidean distance

If we consider that the vector representation of two documents represents a point in the Euclidean space, we could simply take the Euclidean distance between the two points as a measure of similarity. The Euclidean distance between points $\mathbf{p}$ and $\mathbf{q}$ is calculated as:

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \mathrm{d}(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + \cdots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{2.7}$$

The range of the Euclidean distance is the positive real numbers. We normalize the distance by dividing it by the maximum distance possible, which gives us a normalized distance with range $[0, 1]$.

We define the maximum distance possible between two vectors V_0 and V_1 as the Euclidean distance from $V_0 + V_1$ to the origin. In the context of document similarity between documents D_0 and D_1, the maximum distance possible corresponds to the similarity between $D_0 \cup D_1$ and the empty document. Intuitively, it is the maximum distance possible if we were allowed to transfer terms from one document to the other.

2.3 Weighting scheme: tfidf

Tfidf is an acronym for term frequency–inverse document frequency. It measures the importance of a term in a document from a corpus or collection of documents. It is proportional to the number of times the term appears in the document, but the measure decreases as the frequency of the term increases in other documents. This value will only be high if the term appears a large

number of times in the document and the term is not a common term i.e. it does not appear in other documents as much if, at all.

Its calculation is the product of two components, the term frequency and the inverse document frequency:

$$tfidf(t, d, D) = tf(t, d) \times idf(t, D) \tag{2.8}$$

The term frequency is the number of occurrences of the term t in the document d. It could also be a binary value returning 1 if the term occurs in the document and 0 otherwise, in which case we would refer to it as binary TF. The logarithm of the number of occurrences can be also used, but in our work we are going to use the first definition as it is standard.

The inverse document frequency is a measure of how common the term is across all the documents in the corpus. It is the logarithm of the fraction of the documents that contain the term, and it is calculated by dividing the total number of documents by the number of documents that contain the term, and then taking the logarithm of the fraction:

$$idf(t, D) = \log \frac{N}{|d \in D : t \in d|} \tag{2.9}$$

where N is the total number of documents in the corpus and $|d \in D : t \in d|$ are the number of documents where the term t appears.

We can use this weighting scheme with any similarity measure other than the Jaccard index described previouslys.

2.4 Clustering Evaluation Metrics

The task of clustering consists in grouping objects that are similar among them. It differs from the classifying task by the fact that the number of classes is not known beforehand. There are two different kinds of metrics for evaluating the performance of the clustering task. The evaluation metrics are intrinsic when they are based on how close elements from the same cluster are to each other and how distant they are from other clusters. On the other hand, the extrinsic metrics are based on the comparison of the algorithm output with a human built *gold standard* that represents the ideal solution. Many extrinsic methods exist for comparing two clustering outputs, and which of these is best suited for this task remains a subject of debate. In the following sections, we will see a few examples of extrinsic evaluation metrics.

2.4.1 Evaluation by set matching

This first set of metrics was initially identified by [44]. They rely on the concepts of precision and recall from Information Retrieval. Given C as the set of clusters to be evaluated, and L as the set of categories or set by the gold standard, we define purity as:

$$\text{Purity} = \sum_i \frac{|C_i|}{n} max\text{Precision}(C_i, L_j) \tag{2.10}$$

where the precision for a given cluster and category is defined as:

$$\text{Precision}(C_i, L_j) = \frac{|C_i \cap L_j|}{|C_i|} \tag{2.11}$$

Purity rewards not mixing elements from different categories into the same cluster. It does not reward putting elements from the same category into the same cluster. Therefore, we can trivially reach maximum purity by creating one cluster per element.

Inverse purity captures the opposite concept. That is, it does not reward not mixing elements from different categories into the same cluster, but does reward putting elements from the same category into the same cluster. Therefore, we can trivially reach maximum purity by creating one cluster for all elements.

$$\text{Inverse Purity} = \sum_i \frac{|L_i|}{n} max\text{Precision}(L_i, C_j) \tag{2.12}$$

A third measure captures both concepts at the same time. It can be created by combining the concepts of Purity and Inverse Purity using Van Rijsbergen's F measure (harmonic mean) [47]:

$$F = \sum_i \frac{|L_i|}{n} max_j F(L_i, C_j) \tag{2.13}$$

where

$$F(L_i, C_j) = \frac{2 \times Recall(L_i, C_j) \times \text{Precision}(L_i, C_j)}{\text{Recall}(L_i, C_j) + \text{Precision}(L_i, C_j)} \tag{2.14}$$

$$\text{Recall}(L, C) = \text{Precision}(C, L) \tag{2.15}$$

The general formula for the F1 score is:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \tag{2.16}$$

It is not trivial to find a distribution of clusters that optimize the F measure as there is a trade-off between precision and recall. This means that usually a change in the distribution that will increase the precision will make the recall decrease and likewise.

Setting the parameter β allows us to give different relative weights to the partial measures that compose the combined measure.

2.4.2 Metrics based on counting pairs

Another family of metrics is based on counting pairs [45]. For each element, there are 4 possible outcomes: it can belong to the same cluster and category, to the same cluster and different category, to different cluster and same category, or to different cluster and category. Let SS be the number of pairs that fall in the first possible outcome, SD in the second, DS in the third, and DD in the fourth. We can then define the following metrics:

$$\text{Rand statistic R} = \frac{(SS + DD)}{SS + SD + DS + DD} \tag{2.17}$$

$$\text{Jaccard Coefficient J} = \frac{(SS)}{SS + SD + DS} \tag{2.18}$$

$$\text{Folkes and Mallows FM} = \sqrt{\frac{SS}{SS + SD}\frac{SS}{SS + DS}} \tag{2.19}$$

2.4.3 Metrics based on entropy

The entropy of a cluster [40] measures how the members of a category are distributed within clusters. We can take the average over all clusters to get a global measure.

$$\text{Entropy} = -\sum_j \frac{n_j}{n} \sum_i P(i,j) \times \log_2 P(i,j) \tag{2.20}$$

where $P(i,j)$ is the probability of finding an object from category i in cluster j, n_j is the number of objects in cluster j, and n is the total number of objects.

2.4.4 Evaluation metrics based on edit distance

[41] introduces a new family of evaluation metrics that is based on the concept of edit distance. Edit distance measures the minimum number of steps required to transform one partition into another (in other words, transforming the clustering partition into the category partition). For instance, an operation could be to move an object from one cluster to another. Another operation could be to merge two clusters.

2.4.5 B-Cubed F-Measure

The B-Cubed F-Measure computes the precision and recall for each element of the distribution. Precision for an object measures how many objects in the same cluster belong to its category. Similarly, the recall of an object measures how many objects from its category appear in its cluster.

It was first described as an algorithm in [42], but it can also be defined as a function. We define $L(e)$ and $C(e)$ as the category and the cluster, respectively,

of the object e. We can subsequently define the concept of correctness of the relation between e and e' as:

$$\text{Correctness}(e, e') = \begin{cases} 1 & L(e) = L(e') \leftrightarrow C(e) = C(e') \\ 0 & otherwise \end{cases} \tag{2.21}$$

Two objects are correctly related if they belong to the same category if and only if they belong to the same cluster. The B-Cubed precision of an object is then the proportion of correctly related items of that object with all the other objects in its cluster (including itself). The total B-Cubed precision is the averaged precision of each object in the distribution. The B-Cubed recall is analogous to replacing "cluster" with "category".

$$\text{Precision B-Cubed} = Avg_e[Avg_{e' C(e)=C(e')}[\text{Correctness}(e, e')]] \tag{2.22}$$

$$\text{Recall B-Cubed} = Avg_e[Avg_{e' L(e)=L(e')}[\text{Correctness}(e, e')]] \tag{2.23}$$

As in the Purity metrics, the above two measures both have trivial ways of optimizing. Precision will trivially reach its maximum value of 1 when we put each object in a unitary cluster. Similarly, recall will reach its maximum value of 1 when we put all objects into one big cluster. We can combine both measures into one number using the F measure, and it is calculated as follows:

$$F(R, P) = \frac{1}{\alpha(\frac{1}{P}) + (1 - \alpha)(\frac{1}{R})} \tag{2.24}$$

where R and P are two evaluation metrics and α and $(1-\alpha)$ are the relative weight of each metric. $\alpha = 0.5$ leads to an harmonic mean of R and P.

The B-Cubed F-Measure has several properties that make it a good candidate for our project. It uses the concepts of precision and recall, which are easy to interpret. It considers the overall disorder of each cluster, similarly to the entropy based metrics. It also considers the relation between pairs of objects like the metrics based on counting pairs. And most importantly, the B-Cubed F-Measure is the only measure that complies with four very important constraints that we will describe in the next section.

2.4.6 Constraint Satisfaction

In this section, we will describe four constraints that have been selected in [29] as a prerequisite for a good clustering evaluation metric. They all share the properties of being intuitive and easy to understand. It should be possible to demonstrate formally which metrics comply with the constraints, and this would allow us to discriminate metric families. Furthermore, the set of constraints cover all quality aspects that have been proposed in previous work, and have been validated in an experiment involving human assessments.

Constraint 1: Cluster homogeneity

This constraint was first introduced by [43]. Let S be the set of objects belonging to categories $L_1 ... L_n$. We define D_1 as a cluster distribution with one cluster C with objects from two categories L_i, L_j. We define D_2 as D_1 but the cluster C is split into two clusters with objects with category L_i and the objects with category L_j, respectively. Then, the evaluation metric Q must satisfy $Q(D_1) < Q(D_2)$.

Intuitively, cluster homogeneity implies that clusters should not mix items belonging to different categories. This restriction is illustrated in Figure 2.2.

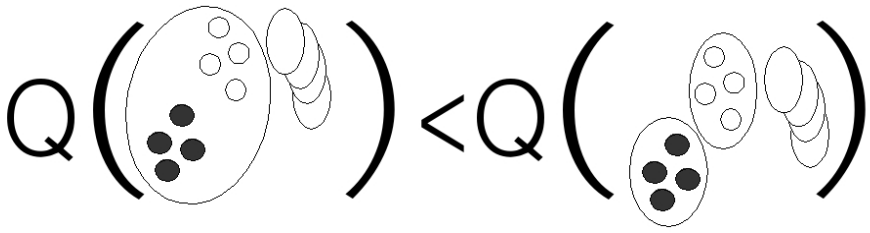


Figure 2.2: Constraint 1: Cluster homogeneity. Source: [29].

Constraint 2: Cluster completeness

Let D_1 be a distribution such that two clusters C_1, C_2 only contain items belonging to the same category L. Let D_2 be an identical distribution, except for the fact that C_1 and C_2 are merged into a single cluster. Then, the evaluation metric Q must satisfy $Q(D_1) < Q(D_2)$.

Cluster completeness is the counterpart to the cluster homogeneity constraint. It enforces that objects belonging to the same category should be grouped in the same cluster. In other words, different clusters should contain items from different categories. This restriction is illustrated in Figure 2.3.

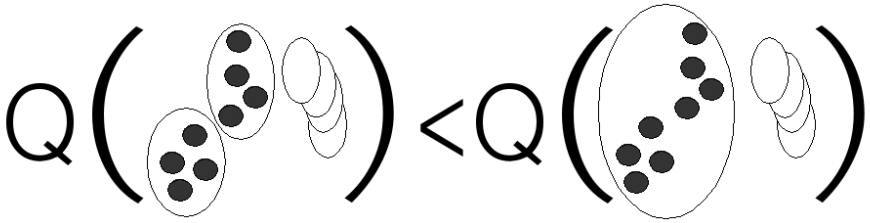


Figure 2.3: Constraint 2: Cluster Completeness. Source: [29].

Constraint 3: Rag Bag

Let C_{clean} be a cluster with n items belonging to the same category. Let C_{noisy} be a cluster merging n objects from unary categories (with just one object per category). Let D_1 be a distribution with a new object from a new category

merged with the cluster C_{clean}, and D_2 another distribution with this new object merged with the cluster C_{noisy}. Then the evaluation metric Q must satisfy $Q(D_1) < Q(D_2)$.

This constraint captures the idea that introducing disorder into a disordered cluster is less harmful than introducing disorder into a clean cluster. In many situations, it is useful to have a cluster with all the non classifiable objects labeled as "miscellaneous", "other" or "unclassified", for example. A perfect clustering system would be able to detect that the rag bag is made out of objects belonging to different classes. In the sub-optimal level, however, this constraint means that we will prefer to have homogeneous clusters plus a rag bag to having less homogeneous clusters. This restriction is illustrated in Figure 2.4.

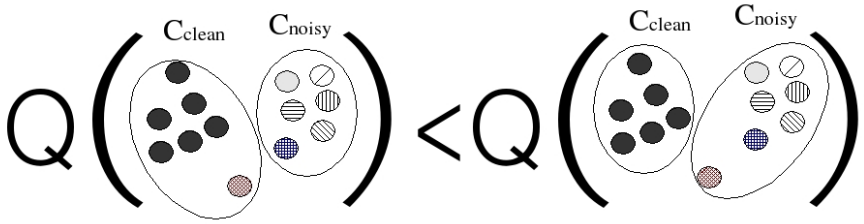


Figure 2.4: Constraint 3: Rag Bag. Source: [29].

Constraint 4: Cluster size vs quantity

Let the distribution D that contains a cluster C_l with $n+1$ objects belonging to the same category L, and n additional clusters $C_1...C_n$, each of which contains two items from the same category $L_1..L_n$. If D_1 is a new distribution similar to D where each C_i is split in two unary clusters; and D_2 is a distribution similar to D, where C_l is split in one cluster of size n and one cluster of size 1; then the evaluation metric Q must satisfy $Q(D_1) < Q(D_2)$.

Intuitively, this constraint implies that we prefer a small error in a big cluster to a large number of small errors in small clusters. This constraint was first introduced in [44]. This constraint likewise implies that we prefer to separate one item from its class of $n > 2$ members to fragmenting n into item categories. Figure 2.5 illustrates this restriction.

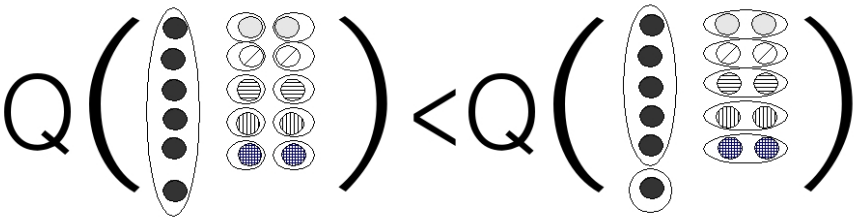


Figure 2.5: Constraint 4: Cluster size vs quantity. Source: [29].

Satisfaction of formal constraints

It can be mathematically proven that the F-measure of the B-Cubed metric is the only metric we described that complies will all the constraints, and this proof is developed in [29]. For this reason, we choose to continue our analysis with the B-Cubed F-Measure.

2.4.7 Unanimous Improvement Ratio

The F-measure has many advantages, but it has a major drawback. It depends on an arbitrary parameter β, small changes to which can lead to very different values of F. To solve this issue, it is useful to complement the F measure with another measure that does not depend on weighting schemes. The Unanimous Improvement Ratio (UIR) has been proven to be the only measure that fulfills this requirement [68]. For a number of test cases, and for two different runs S_1 and S_2, the UIR of S_1 against S_2 is calculated as:

$$\text{UIR}(S_1, S_2) = \frac{N_{>\forall}(S_1, S_2) - N_{>\forall}(S_2, S_1)}{\text{Amount of cases}} \tag{2.25}$$

where $N_{>\forall}(S_1, S_2)$ is the number of test cases for which $S1$ improves $S2$ for all measures.

Chapter 3

Related work

In this chapter, we will review the research conducted on topic detection in short texts, which nonetheless remains an open problem, as well as present the different approaches proposed in the literature.

We can group the existing approaches into three groups: LDA-based algorithms, Clustering algorithms, and other. While our approach falls into the Clustering algorithms category, this chapter provides a more general overview of the approaches.

3.1 LDA-based topic models

The LDA (Latent Dirichlet allocation) model is an unsupervised machine learning technique which uncovers information about latent topics across the corpora. It assumes that documents are represented as a probability distribution over topics, and each topic is represented as a probability distribution over terms. The method infers the hidden topics with their term distributions and topic distributions for each text from the document texts alone. In this model, each document is represented as a bag or multiset (set with a repetition count) of its terms, i.e. terms are exchangeable.

LDA defines the following generative process for each document in the collection:

- For each document, pick a topic from its distribution over topics.
- Sample a term from the distribution over the terms associated with the chosen topic.
- The process is repeated for all the terms in the document.

This method has been successfully used for clustering large or medium sized documents such as webpages, articles or books. With respect to short text clustering such as tweets, however, this methodology has been proven not to

perform as well [79]. Therefore, modifications of the model have been proposed for this particular task.

To overcome the difficulty of sparse information related to short texts, studies proposed to aggregate all the tweets of a user into a single document. This treatment can be regarded as an application of the author-topic model to tweets, where each document (tweet) has a single author.

3.1.1 The Author-Topic Model

The Author-Topic Model (AT model) is an extension of LDA. In this model, each term w in a document is associated with two latent variables: an author x and a topic z. Each author in the collection is associated with a multinomial distribution over T topics, and each topic is associated with a multinomial distribution over terms. In contrast to LDA, the observed variables for an individual document are the set of authors and the terms in the document. The formal generative process of the AT model is as follows:

- For each document, given the vector of authors.
- For each term in the document, conditioned on the author set a_d , choose an author $x_{di} \sim Uniform(a_d)$.
- Conditioned on x_{di}, choose a topic z_{di}.
- Conditioned on z_{di}, choose a term w_{di}.

Example 1

One example is [80]. This study focuses on the problem of identifying influential users of micro-blogging services. TwitterRank, an extension of PageRank algorithm, is proposed to measure the influence of users in Twitter. TwitterRank measures the influence taking both the topical similarity between users and the link structure into account. For this purpose it is needed to automatically identify the topics that authors are interested in based on the tweets they published, and the LDA model is applied. In LDA documents should naturally correspond to tweets, but since the goal is to infer the topics that each author is interested in rather than the topic of each single document, the documents published by an individual author are aggregated into a big document. Thus, each document essentially corresponds to an author. Experimental results show that TwitterRank outperforms the one Twitter currently uses and other related algorithms, including the original PageRank and Topic-sensitive PageRank.

Example 2

In another study [81] several schemes to train a standard topic model are proposed and their quality compared from both qualitative and quantitative perspectives.

- Train LDA using messages
- Train LDA using aggregated user profiles with combined messages from the same user
- Train LDA using aggregated term profiles with combined messages that contain that term
- Train extended AT model

The AT model is extended to allow each message to have a fictitious author who is indeed the message itself. Thus, for each message, terms are either sampled from the author specific topic mixture or from the message specific topic mixture.

It is shown that the effectiveness of trained topic models can be highly influenced by the length of the documents; namely, a better model can be trained by aggregating short messages.

Moreover, through the experiments, it is shown that the simple extension to the AT model does not yield better modeling for messages and users and indeed it is worse than training a standard LDA model on user aggregated profiles

3.1.2 Twitter-LDA

Example 1

Aggregating all documents from the same author into a big document does not exploit the fact that a single tweet is usually about a single topic. In [79] the Twitter-LDA model is proposed. In this model it is assumed that there are T topics in Twitter, each one represented by a term probability distribution. Each topic and the background terms are represented by a term distribution. Each user is represented as a topic distribution. The generative process of Twitter-LDA model is as follows: A Bernoulli distribution governs the choice between background terms and topic terms. When topic terms are selected, a topic is chosen from his topic distribution. Finally a bag of terms is chosen one by one based on the selected topic or the background model.

The effectiveness of the Twitter-LDA model is quantitatively evaluated and compared with the standard LDA model and the author-topic model. It is shown that the Twitter-LDA model clearly outperforms the other two models, giving more meaningful top topic terms, indicating that the Twitter-LDA model is a better choice than the standard LDA for discovering topics on Twitter.

Example 2

In another study [21] an approach inspired by the TwitterLDA model [65] and TOT model [64] is proposed. In the TOT model each topic is assigned to a continuous distribution over time. For each document, the distribution over topics depends on the term co-occurrences and the tweet creation time. In both the TwitterLDA and TOT models tweets are about one single topic and each

topic has a time distribution. In this approach the following assumptions are made: (a) For each entity of the dataset, there is a set of topics K, and each topic is represented by a term distribution and a continuous distribution over time. (b) When a tweet is written, the author chooses if its going to be a background term or a topic term. In the former case, the author chooses a term from the term distribution of the topic. In the later case, the author chooses a topic based on the topic distribution for the entity and then chooses a term from the topic. The process is described in the following pseudo-code:

1. Draw $\theta_d \sim Dir(\alpha), \phi_{\mathcal{B}} \sim Dir(\beta), \pi \sim Dir(\gamma)$
2. For each topic $z = 1, ..., K$
   (a) draw $\phi Z \sim Dir(\beta)$
3. For each tweet $d = 1, ..., D$
   (a) draw a topic $z_d \sim Multi(\theta_d)$
   (b) draw a timestamp $t_d \sim Beta(\psi_{z_d})$
   (c) for each term $i = i, ..., N_d$
      i. draw $y_{d_i} \sim Bernoulli(\pi)$
      ii. if $y_{d_i} = 0$: draw $w_{d_i} \sim Multi(\phi_{\mathcal{B}})$
      if $y_{d_i} = 1$: draw $w_{d_i} \sim Multi(\phi_{z_d})$

where ϕ_{z_d} is the term distribution for topic z, ψ_{z_d} is the time distribution for the topic z, $\phi_{\mathcal{B}}$ is the term distribution for background terms, and π is a Bernoulli distribution used to choose between background terms and topic terms. Gibbs sampling is used to perform the inference. The graphical model is shown in the following figure.

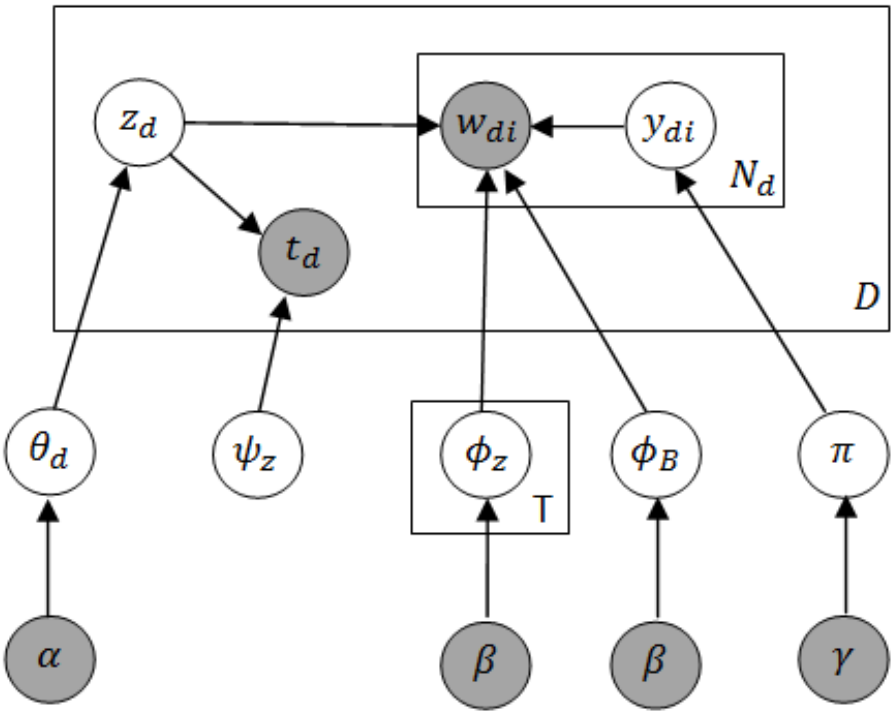


Figure 3.1: Graphical model for the LDA-based clustering. Source: [21]

After applying the model, each topic will be represented as a distribution over the vocabulary, and each tweet will be assigned to a single topic.

The system uses background data (unlabeled tweets) to improve the clustering. The system is able to adapt to find the right number of clusters, thus overcoming one of the limitations of LDA based systems, that is, the need to know a priory the number of clusters.

This approach performs much better than the baselines in the 2013 edition of the Replab competition, and performs better when the background collection used for transfer learning contain tweets from different entities/test cases.

3.1.3 Labeled-LDA

Example 1

Recently, [82] applied Labeled-LDA to Twitter. Labeled-LDA extends LDA by incorporating supervision with implied tweet-level labels, enabling explicit models of text content associated with hashtags, replies or emoticons. Unfortunately the model relies on hashtags, which may not include all topics.

Labeled LDA allows to model a collection of Twitter posts as a mixture of some labeled dimensions as well as the traditional latent ones like those discovered by LDA. Labeled LDA assumes the existence of a set of labels, each

characterized by a multinomial distribution over all terms in the vocabulary. The model assumes that each document d uses only a subset of those labels, and that document d prefers some labels to others as represented by a multinomial distribution. Each term in each document is picked from a term distribution associated by that document's labels.

LDA is a special case of Labeled LDA. If we model topics as labels and no other labels are used, then Labeled-LDA is mathematically identical to traditional LDA.

In this study [82] it is shown that LDA related models work on documents as short as Twitter posts, and that these methods can support rich analyzes of Twitter content at large scale and at the level of individual users with mapping sets of posts into substance, status, social, and style dimensions.

3.2 Clustering algorithms

Clustering algorithms group together objects that are similar among them based on a particular similarity measure. We differentiate two groups: Hierarchical clustering, and Network community detection. In the following sections we will describe them and give some examples.

3.2.1 Hierarchical clustering

Hierarchical clustering (HCA) is a method of cluster analysis that generates a hierarchy of clusters. The results are usually presented as a dendrogram. There are two main strategies: (a) Agglomerative: a bottom up approach where the initial state has each element in its own cluster and clusters are merged as the algorithm progresses. (b) Divisive: a top down approach where all elements start in one big cluster that is recursively split. In general this kind of algorithms are greedy algorithms.

Example 1

In the 2012 edition of the Replab competition for Online Reputation Management systems [31], a Hierarchical Agglomerative Clustering algorithm is used as the baseline clustering algorithm from which other algorithms compare.

The Jaccard distance is used as the document similarity measure. The threshold is defined at various levels (0,10,20,...,100) resulting in 11 different versions of the baseline. Although it is not specified in the paper, we assume that since the similarity measure used is the Jaccard distance, the document representation is a set of terms.

Surprisingly, the best performing algorithm of the competition for the clustering task is the baseline with threshold 0, which is equivalent to the trivial algorithm of clustering all elements into one big cluster, showing that the competing systems were not contributing to solve the problem.

Example 2

We can see another example of a Hierarchical Agglomerative Clustering algorithm in [21]. The idea of wikified tweets is to link each tweet with one or several entities defined in a knowledge base, such as Wikipedia. It exploits the hypothesis that two tweets sharing concepts will be likely to be related and therefore should be classified with the same topic. To find the entities linked to a tweet, the Commonness probability [62] is used that calculates the probability of an n-gram q to be linked to a Wikipedia article c. For each of the longest n-grams of the tweet, the article with higher Commonness probability is chosen.

$$\text{Commonness}(c, q) = \frac{|L_{q,c}|}{\sum_{c'} |L_{q,c'}|} \tag{3.1}$$

where $L_{q,c}$ is the set of all links in Wikipedia with anchor text q and target c. Both the Spanish and English Wikipedias are used depending on the language used in the tweet. When the Spanish language is used, the Wikipedia articles found linked to the tweet are translated to English following the inter-lingual links using the Wikimedia API. After the wikifying the tweets, the tweets are represented as a set of Wikipedia articles, and the Jaccard similarity measure is used to compare the tweets using the set of articles linked to them. Finally a agglomerative clustering approach is used to cluster the tweets into categories. Two tweets $C1$ and $C2$ will be assigned into the same category if the Jaccard similarity of their assigned entities $C1$ and $C2$ is above a previously specified threshold $\text{Jaccard}(C1, C2) > th$.

The wikified tweet clustering gets the highest score in all the metrics in the 2013 edition of the Replab competition.

Example 3

In [21] the terms appearing in the tweets are classified into categories, and later the tweets are classified on those categories.

For the first step, a Hierarchical Agglomerative Clustering approach is used to build the categories. For that, the confidence score given by a classifier is used as the similarity function. The classifier tries to guess whether two terms belong to the same category or not. The selected input data for the classifier is the following:

- **Term features**: For each term of the pair, the following features are considered: term occurrence, normalized frequency, pseudo-document TF.IDF and KL-Divergence [63]. The features are computed in two ways, by using only the training set or by using both the training set and the background data. Tweet metadata is also used: Shannon's entropy of named users, URLs, hashtags and authors in the tweets where the term occurs.

- **Content-based pair term features**: These are the features that are calculated using both terms: Shannon's entropy of named users, URLs, hashtags and authors in the tweets where the term occurs.

- **Meta-data-based pair term features**: These are the features that are calculated using both terms metadata: Jaccard similarity and Shannon's entropy of named users, URLs, hashtags and authors between tweets where both terms co-occur.

- **Time-aware features**: These are the features that use the date of creation of the tweets: median, minimum, maximum, mean, standard deviation, Shannon's entropy and Jaccard similarity. They are calculated using four different time intervals: milliseconds, minutes, hours and days.

For the classification model, each pair of terms in the training set for the classifier need to be labeled as *clean* or *noisy*, where clean means that the two terms are very likely to belong to the same topic, whereas noisy means that the probability is lower. A pair of terms are classified as clean when 90% of the tweets belong to the same cluster, otherwise they are classified as noisy. More formally:

$$\text{label}(< t, t' >) = \begin{cases} \text{clean} & \max_j \text{Precision}(C_{t\hat{t'}}, L_j) > 0.9 \\ \text{noisy} & \text{otherwise} \end{cases}$$

where $C_{t\hat{t'}}$ is the set of tweets where the terms t and t' co-occur, and L is the set of topics in the goldstandard, and

$$\text{Precision}(C_i, L_j) = \frac{C_i \hat{L}_j}{C_i} \tag{3.2}$$

A binary classifier is built using the training set previously constructed. The confidence output of the classifier for the classification of clean or noisy is used as a similarity measure of the two input terms. This similarity measure is used to build a similarity matrix that is the input of a Hierarchical Agglomerative algorithm that classifies the terms into categories. The cut-off threshold chosen is based on the number of possible merges is used to return the final term clustering solution.

The second and final step is two classify each tweet into the categories constructed in the previous step. This is accomplished by assigning each tweet to the category represented with the set of terms that have a higher Jaccard similarity with the terms of the tweet.

3.2.2 Network community detection

Network community detection algorithms aim to detect the community structure of a network. We can define a network with nodes as the documents and edges weighted according to the similarities of the documents that are linking. In this case a network community detection algorithm will effectively find clusters of similar documents.

Example 1

This study [58] uses classical measures for the document similarity and community detection techniques for topic detection. The similarity matrix is constructed by using the cosine similarity with a tfidf weighting scheme. No distinction is made over the two possible languages. A simple s-stemmer is applied to all terms. All terms with less than 4 characters are removed. Once the similarity matrix is produced, the communities are found by using the VOS community detection algorithm [23]. Although it is not mentioned in the study, we assume that the tweet representation is a bag of terms given that the cosine similarity is used. For each of the communities found, their density is calculated. The density of a network or subnetwork is the sum of the weights of all the links over the maximum number possible. More formally, the density for the network represented for the similarity matrix (with values ranging from 0 to 1) A is:

$$\text{density}(A) = \frac{\sum_i \sum_j A_{ij}}{\sum_i \sum_j 1} \tag{3.3}$$

In another version of the algorithm, a filter is applied where only communities above a certain threshold are assigned a category, communities below the threshold are left without a category or with its own category. The performance of the second version (with density filter) of the algorithm is better than most of its competitors as it is shown in the ranking in [58]. The original version of the algorithm (without density filter) performs more poorly. This result is consistent across different ways of measuring it.

Example 2

Another example of a network community detection approach for topic detection is twitterstand [6]. It is an online clustering algorithm based on the leader-follower clustering [8]. The leader-follower algorithm is based upon identifying the internal structure of the expected communities. It uses the notion of network centrality to differentiate leaders (nodes which connect different communities) from loyal followers (nodes which only have neighbors withing a single community). This algorithm is able to learn the communities from the network structure, and it does not require knowledge of number of communities in the network as it learns it naturally.

Since tweets are coming at a high throughput rate, TwitterStand needs to be very efficient. Once a tweet is clustered, it will stay there forever. The original leader-follower algorithm is modified so that it works in an online fashion and is more resilient to noise. The clustering algorithm keeps a list of active clusters, represented with a (tfidf) weighted list of terms collected from the contained tweets. For each cluster, the time centroid or the mean publication time of all tweets forming the cluster is also stored. A cluster becomes inactive when the centroid is older than three days. Each input tweet t is represented by its feature vector representation using tfidf. A variant of the cosine similarity measure is used for computing the distance between t and the candidate cluster.

A Gaussian attenuator on the cosine distance is used so that that it favors input tweets being added to those clusters whose time centroids are close to the tweet's publication time. In particular, the Gaussian parameter takes into account the difference in days between the cluster's time centroid and the tweet's publication time.

Although no results are shown on this study, an online version of the system can be used at [7].

3.3 Other topic modeling

In this section we will show example algorithms that do not fall into any of the previous categories.

Example 1

In [20] the topic detection task relies on the identification of headwords (HW) characteristic of each topic. HW are terms, bigrams, distance bigrams and tweet author that are selected using a maximum a posteriori probability estimator. For each topic, a list of HW is created and ranked based on a purity criterion and an initial set of discriminative theme headwords HW_k is selected. The size of all HW vocabularies has been set to be equal across topics. Some headwords may appear in more than one topic vocabulary.

In order to assign a topic to a tweet, the topic contribution of the tweet Y_d is computed per each topic T. The topic contribution $HW(T_k|Y_d)$ is the sum of the contributions of the tweet in the features selected for that topic. The tweet is assigned to the topic with maximum HW contributions.

In [24] we can see that this proposal ranks number three in the 2013 edition of the Replab competition.

Example 2

TweetMotif [4] is an exploratory search application for Twitter that groups messages by frequent significant terms — a result set's subtopics — to facilitate navigation. The topic extraction system is based on syntactic filtering, language modeling, near-duplicate detection, and set cover heuristics.

Instead of simply showing this result set as a list, TweetMotif extracts a set of topics to group and summarize these messages. A topic is simultaneously characterized by (1) a 1- to 3-term textual label, and (2) a set of messages, whose texts must all contain the label. TweetMotif takes a simple language modeling approach to identifying topic phrases that are most distinctive for a tweet result set, scoring them by the likelihood ratio:

$$\frac{Pr(phrase|tweetresultset)}{Pr(phrase|generaltweetcorpus}$$

Every candidate phrase defines a topic, a set of messages that contain that phrase. Many phrases, however, occur in roughly the same set of messages, thus their topics are repetitive. Therefore similar topics are merged.

There are no results shown in the study, but an online version of the system can be found at [5].

Example 3

The baseline of the 2013 edition of the Replab competition consists in tagging every tweet from the test set with the same tag as the closest tweet in the training set. The similarity measure used is the Jaccard coefficient. This methodology is easy to understand, can be applied to many tasks, and it exploits the training set per entity.

In [24] we can see that this proposal ranks number six in the 2013 edition of the Replab competition.

3.4 Summary

In this chapter we have reviewed the state of the art for the short text topic detection task. We have focused on the proposed algorithms, similarity measures and tweet representations. In table 3.1 we give a summary of all of them.

Modularity maximization is the most widely used method for community detection (and thus clustering), yet, to our knowledge, it is still an unexplored area of research. The aim of this study is to fill this gap.

Category	Example	Algorithm	Similarity measure	Document representation
LDA/Author-Topic Model	Example 1	Author-Topic	Not applicable	Set of terms
LDA/Author-Topic Model	Example 2	Author-Topic	Not applicable	Set of terms
LDA/Twitter-LDA	Example 1	Twitter-LDA	Not applicable	Set of terms
LDA/Twitter-LDA	Example 2	Twitter-LDA	Not applicable	Set of terms
LDA/Labeled-LDA	Example 1	Labeled-LDA	Not applicable	Set of terms
Clustering/Hierarchical	Example 1	Hierarchical Agglomerative Clustering	Jaccard	Set of terms
Clustering/Hierarchical	Example 2	Hierarchical Agglomerative Clustering	Jaccard	Set of Wikipedia articles
Clustering/Hierarchical	Example 3	Hierarchical Agglomerative Clustering	Jaccard	Set of terms
Clustering/Network community detection	Example 1	VOS	Cosine with tfidf	Set of terms
Clustering/Network community detection	Example 2	Leader-follower	Variant of cosine	Feature vector representation using tfidf
Other	Example 1	Max	Topic contribution *HW*	Set of terms
Other	Example 2	Statistical language model	Likelihood ratio	N-grams
Other	Example 3	Max	Jaccard	Set of terms

Table 3.1: Summary of short text topic detection approaches

Chapter 4

Design and Implementation of the proposed system

This chapter describes a high level design and implementation of our system. The first section describes the tweet representation, the second section defines the new similarity measure used in one of our proposed approaches, the third defines a generalization of the modularity measure. Finally, the fourth section presents the implementation details of the program and database.

4.1 Tweet representation

In this section, we will present two different methods for enriching the tweets. Each method corresponds to different approaches, and so they will not be used simultaneously. One method involves following the links included in the tweet in order to obtain the parsed html of the referenced website. This extra information can be used when calculating the tweet similarities and thus avoiding the short-text problem. Another methodology will reference the tweet with Wikipedia entities. Since we can calculate the similarity across Wikipedia entries, this will enable us to calculate the similarity between tweets with our proposed measure, the Fuzzy Weighted Jaccard described later in this chapter.

4.1.1 Html enrichment

Some tweets contain one or more links to websites. We can therefore follow the links to extract extra information that can be useful for clustering. We propose two methodologies for enriching the tweets with html. One is to enrich the tweet with relevant terms found in the html. We propose to use the title of the website because it is likely to be a very short summary of the contents of the website. Another possibility is to weight the terms found in the parsed html code using the tfidf measure, calculating the idf using a corpus with all the html documents in our dataset. We can then choose the top n terms by tfidf to be

included in our tweet. We can use a combination of the above two methods to enrich our tweets.

A different methodology that we propose is to cluster the html documents directly. The problem is that not all tweets provide a link to an html document. We propose to cluster the tweets with html separately from the tweets without. This choice has the drawback that tweets without html links will be separated from the ones with html. On the other hand, it has the benefit of allowing us to cluster the tweets with html with larger documents, which would probably lead to better results by eliminating the short-text problem for those tweets with html.

4.1.2 Wikified tweet

In order to use the Fuzzy Weighted Jaccard semantic similarity measure (defined in section 4.2.1), we need to be able to define a similarity across terms. We propose to wikify our tweets, as this approach has been used in previous works [21]. After the process of wikifying, each term will be assigned a Wikipedia page. Since we can calculate the similarity of two Wikipedia pages (defined in section 4.2.2), the process of wikifying will enable us to calculate the similarity between two terms by calculating the similarity of their assigned Wikipedia pages. We have chosen to use the Wikipedia instead of other encyclopedias or dictionaries such as wordnet because Wikipedia has been successfully used to cluster tweets in previous works [21].

As done in other studies, we will use the Spanish Wikipedia for tweets in Spanish and find the English version of the article using the Wikimedia API.

4.2 Similarity

Our system will use a number of similarity measures previously described in Chapter 2, Background. In this section, we define a new similarity measure that allows to do comparisons at the semantic level of the documents.

4.2.1 Fuzzy Weighted Jaccard

The Fuzzy Weighted Jaccard is a new Fuzzy Jaccard definition that uses weights for its terms. Let A and B be two sets of objects, and let S be the similarity matrix that contains the similarity for each of the objects in A and B. The Fuzzy Weighted Jaccard measure can then be calculated as follows:

$$FWJaccard(A,B) = \max_{x} \frac{\sum_{i \in A} (w_i + w_{x_i}) \times S_{i,x_i}}{\sum_{i \in A \cup B} w_i} \tag{4.1}$$

where x is a mapping between objects of A to objects of B, and w_i is the weight of the object i.

The trivial algorithm that computes the Fuzzy Weighted Jaccard similarity measure involves trying every possible combination of mappings (every possible

x) and giving the maximum value found for FWJaccard. This algorithm has an exponential time complexity that renders it impractical. Instead, we propose a greedy algorithm that despite not guaranteeing to find the maximum possible value of FWJaccard, likely finds a value near to the maximum. It is an iterative process where in each iteration, we find a mapping from an object from A i to an object of B j so that the expression $(w_i + w_j) \times S_{i,j}$ is maximized. We can find this with time complexity O($|A| \times |B|$). Once we find the optimal mapping, we delete the objects i and j from A and B, respectively, and we iterate again. The time complexity of the full algorithm is O($|min(A, B)|^2 \times |max(A, B)|$).

We extend the previous definition by introducing a constant f that allowS us to control the fuzziness of the measure.

$$FWJaccard(A, B) = \max_x \frac{\sum_{i \in A} (w_i + w_{x_i}) \times \text{factor}(S_{i,x_i}, f)}{\sum_{i \in A \cup B} w_i} \tag{4.2}$$

where factor(a,b) is defined as 1 when $a = 1$ and $a \times b$ otherwise. If we use $f = 0$, then the Fuzzy Weighted Jaccard is equivalent to the Weighted Jaccard. For bigger values of f, the fuzzy part of the equation becomes more important.

Let us consider an example of the comparison of two tweets illustrated in figure 4.1.

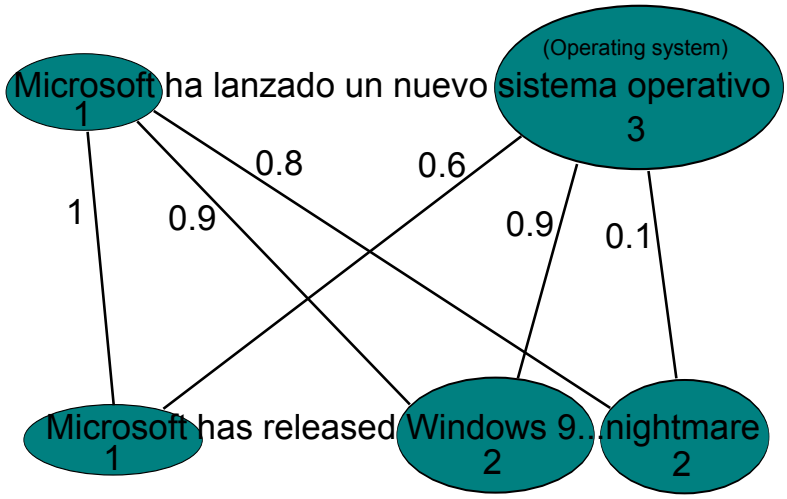


Figure 4.1: Fuzzy Weighted Jaccard example 1

In the first iteration, we would find the highest similarity to be the one between the n-grams *Microsoft* and *Microsoft*. By definition, the similarity

between an object and itself is the maximum possible. Then, both terms are matched and all links that were connecting them are removed. This leaves us with only two links to choose from with weights 0.9 and 0.1, as we can see in figure 4.2.

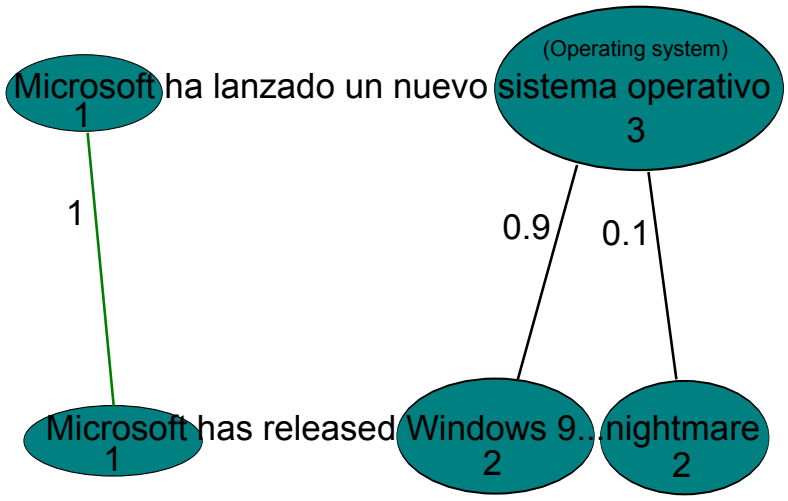


Figure 4.2: Fuzzy Weighted Jaccard example 2

We choose the edge with the highest weight that connects the Spanish n-gram *sistema operativo*, which translates to English as *operating system*, and the n-gram *Windows 9*. We again match these two n-grams and remove the links connecting them. At this point, there are no more links left, and the matching process is finished. We can see the resulting match in figure 4.3.

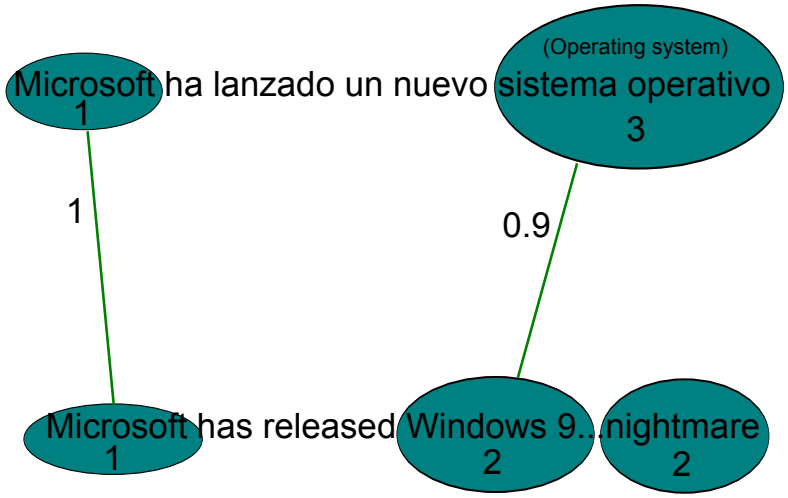


Figure 4.3: Fuzzy Weighted Jaccard example 2

The link weighted Jaccard measure is then calculated as follows:

$$\frac{(1+1)\times 1+(3+2)\times 0.9}{1+1+3+2+2}=\frac{6.5}{9}=0.72$$

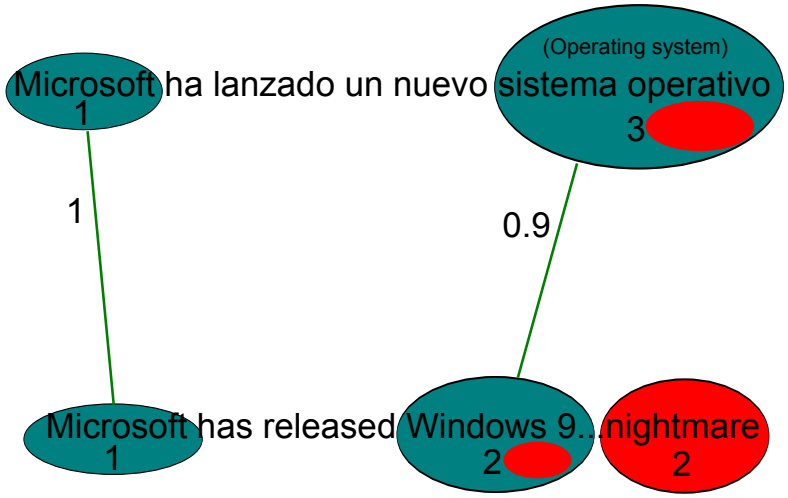


Figure 4.4: Fuzzy Weighted Jaccard example 4

Figure 4.4 demonstrates an intuitive geometrical interpretation of the Fuzzy Weighted Jaccard similarity. The area of the red mark inside the green bubbles represents 10% of the total area of each of the bubbles, as they are connected with similarity 0.9. If a bubble is disconnected, then it is completely red. Then the Fuzzy Weighted Jaccard similarity corresponds to the proportion of green area over the total area.

4.2.2 Similarity between Wikipedia entries

The similarity measure that we will use to compare Wikipedia articles is the classic cosine tfidf of the output links of the articles. We select the output links instead of all the terms in the article for the following reason. Since the number of links is lower than the number of terms, it is faster to calculate the similarity. Efficiency in this calculation is crucial as it will need to be repeated for every possible pair of Wikipedia articles in our set of tweets. For a set of n tweets with an average of m Wikipedia articles per tweet, we are going to compute a total of $n^2 \times m^2$ Wikipedia articles similarity calculations. We choose the links rather than terms for making the comparison because usually all keywords–which are the most relevant terms–from a Wikipedia article are provided with an output link.

This similarity measure will be used to calculate the weights of the edges for the Fuzzy Weighted Jaccard measure in figures 4.1 to 4.4.

4.3 Clustering: extended modularity

We propose to use what we call the extended modularity, which is a generalization of the classic modularity (eq 2.1) that introduces a new parameter α. The parameter α allows us to tune the granularity of the communities found by the optimization algorithm. When α is 1 the extended modularity is equivalent to the classic modularity.

$$Q = \sum_{vw} \left[\frac{A_{vw}}{2m} - \alpha \times \frac{k_v * k_w}{(2m)(2m)} \right] \delta(c_v, c_w) \tag{4.3}$$

The modularity measure has two components: one can be interpreted as the probability that a randomly chosen edge will connect nodes from the same partition, and the second–substracted–component is the same expected probability for the graph after if it were randomly rewired. The second component is important because without it, the optimization algorithm would find the trivial solution of creating a big cluster with all nodes in it. By multiplying the second factor by a parameter α, we can control the granularity of the clustering solution found by the optimization algorithm. Networks tend to have communities and communities inside communities. We can select the level of sub-communities that we want to find by changing the value of α. For instance, for $\alpha = 0$, the algorithm will find one big community with all nodes. Small values of α will tend to give big communities, while bigger values of α will tend to find smaller communities. If the value of α is big enough, the solution found will be one cluster per node.

Figures 4.5 and 4.6f illustrate these ideas.

We use our training set to find the optimal α, and we use this value for the testing set.

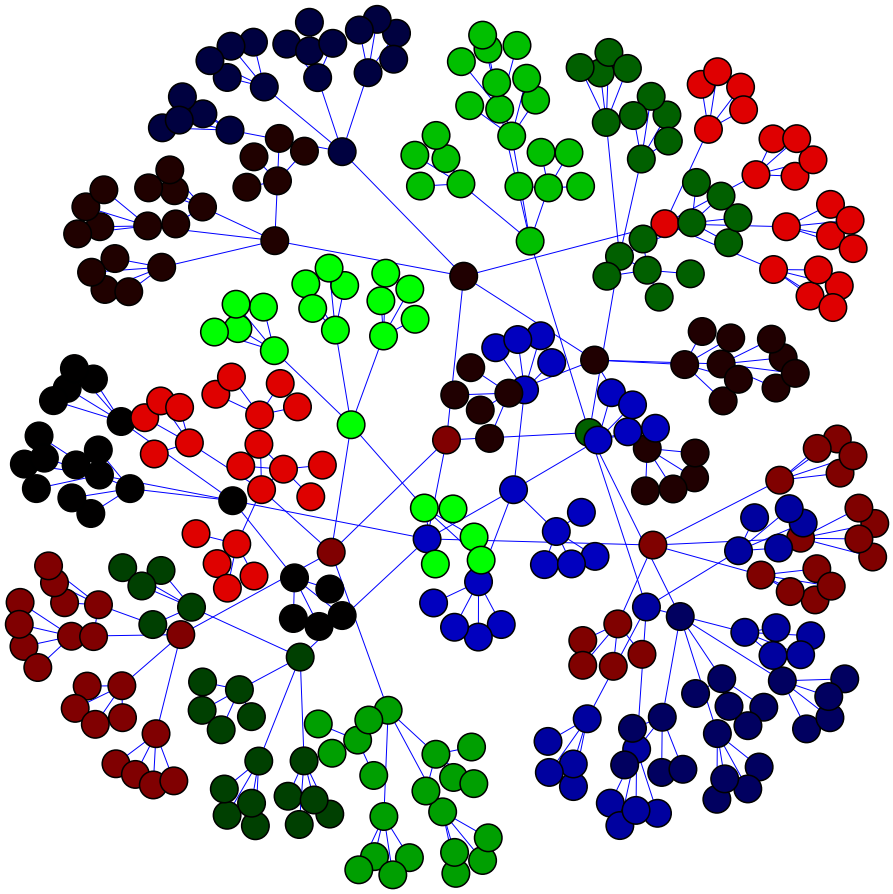
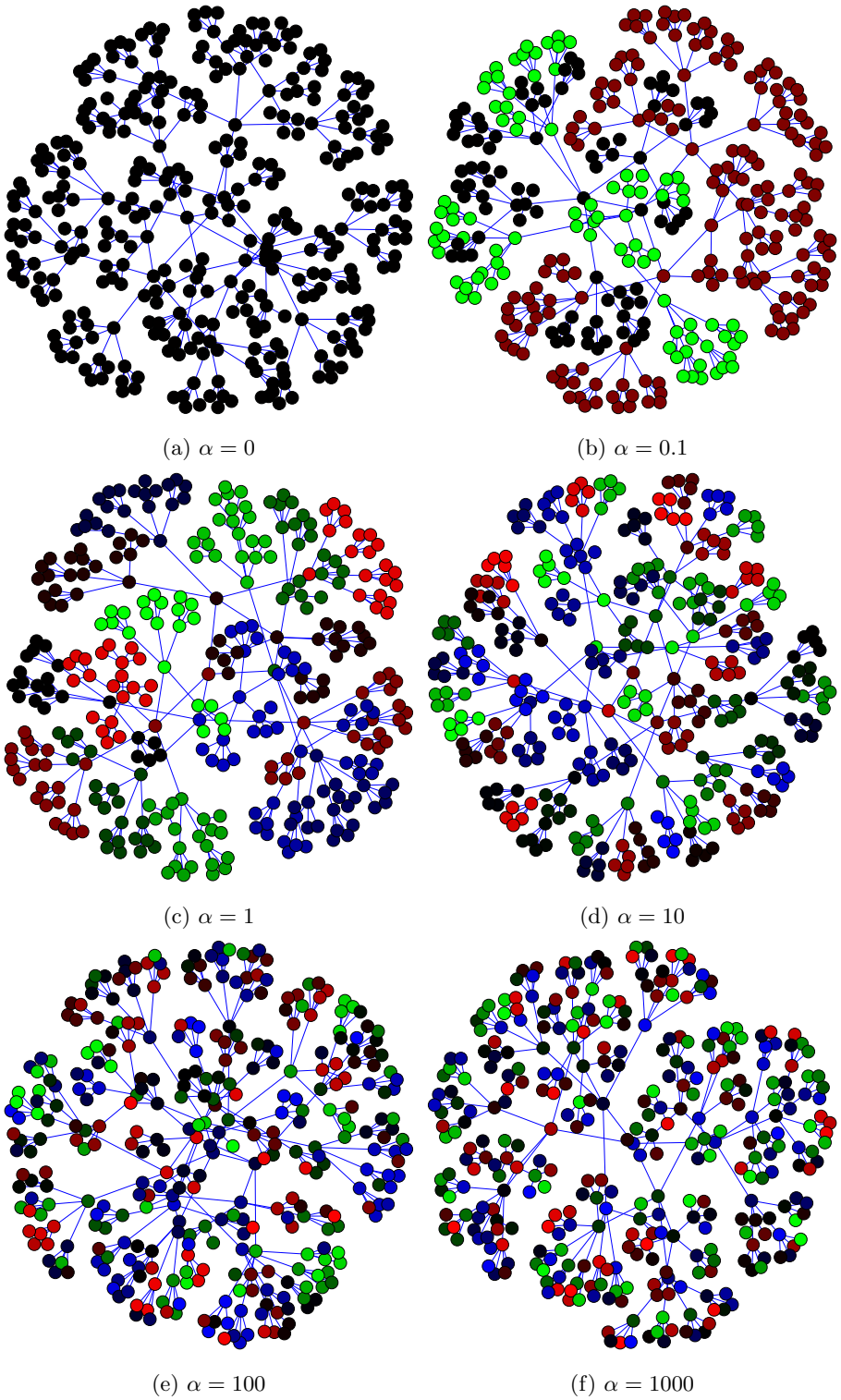


Figure 4.5: Community detection for sample network with $\alpha = 1$



(a) $\alpha = 0$ (b) $\alpha = 0.1$

(c) $\alpha = 1$ (d) $\alpha = 10$

(e) $\alpha = 100$ (f) $\alpha = 1000$

Figure 4.6: Community detection with different values of α

4.4 Implementation

The project has 5,454 lines of Python code. The project is separated into several modules. Figure 4.7 exhibits a diagram of the Python modules with their interactions. The module replab is responsible for accessing the replab datasets. It loads the tweets and optionally loads the html documents to which they link. Loading quickly is accomplished by the object serialization technology *pickle* in Python.

Not all links are html webpages. To ensure that we work with only html webpages, the module uses the file unix command to discard binary files and everything that is not html. Also, the issue of files being encoded in latin-1 and others in utf-8 is handled by trying to load as utf-8 first, and in case of an exception, loading it as latin-1. The module similarity contains all the similarity measures implementations, and it is responsible for generating the similarity matrix. The Wikipedia module gives access to the Wikipedia functions that we will describe in the next section. The louvain module implements the Louvain method. It has been developed by a third party [71] and released under the BSD license. We have modified it to implement the extended modularity. Finally, the main program uses the above-mentioned modules to do all the calculations and generate the tables in Latex. Every run configuration is encoded in one string of comma separated parameters. The results are cached in a file so that re-running a configuration does not require recomputing.

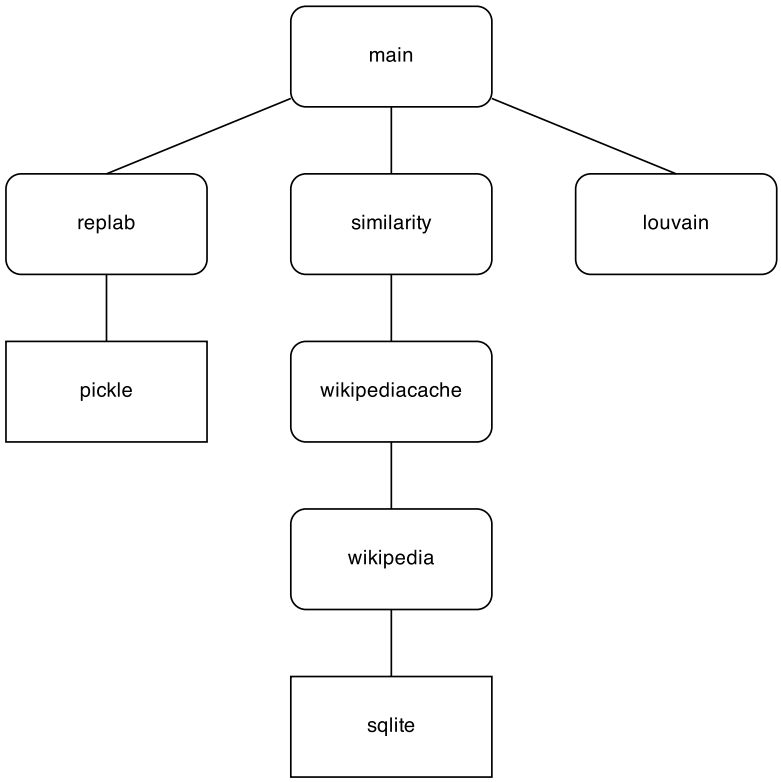


Figure 4.7: Python modules diagram. Rectangles represent standard modules.

4.4.1 Wikipedia database

The task of tweet wikifying is the most computer intensive. For this reason, it requires careful design.

We need to be able to make queries to quickly find all links in the whole Wikipedia for a given anchor. In light of this, we have constructed a database using sqlite with all the links in the Wikipedia. When populating the database using the available database dumps, the anchors are converted to lowercase. This database contains indexes enabling us to make very quick queries at the expense of hard disk space. Databases are constructed for each language separately. A recursive algorithm iterates through all possible n-grams from the tweet starting with the longer n-grams and queries the database matching anchors. If an anchor is found, the most frequent Wikipedia article is selected and the n-gram is marked as used. When the end of the string is found, no more n-grams are possible, and the method recursively calls itself to find (n-1)-grams. The process finishes after we have scanned all 1-grams or we have marked all the terms from the tweet. This algorithm uses the database by using the method wikify(n-gram) that returns one link or nothing.

If the tweet is in Spanish, the articles found are translated using the wikimedia API. This function is provided by the method translate(article).

We provide cached versions of the methods wikify(n-gram) and translate(article), which use another table to store previous calls. This speeds up the process as many terms and n-grams are very frequent. It also allows us to rerun the algorithms faster after the first execution.

In order to calculate the similarity of articles, we need random quick access to the article contents given its article name or link. In view of this, we create another database table that contains an association between articles names or links and articles contents. The database is populated once from the Wikipedia dump, and accessed quickly with indexes when needed.

Figure 4.8 presents a diagram for the database.

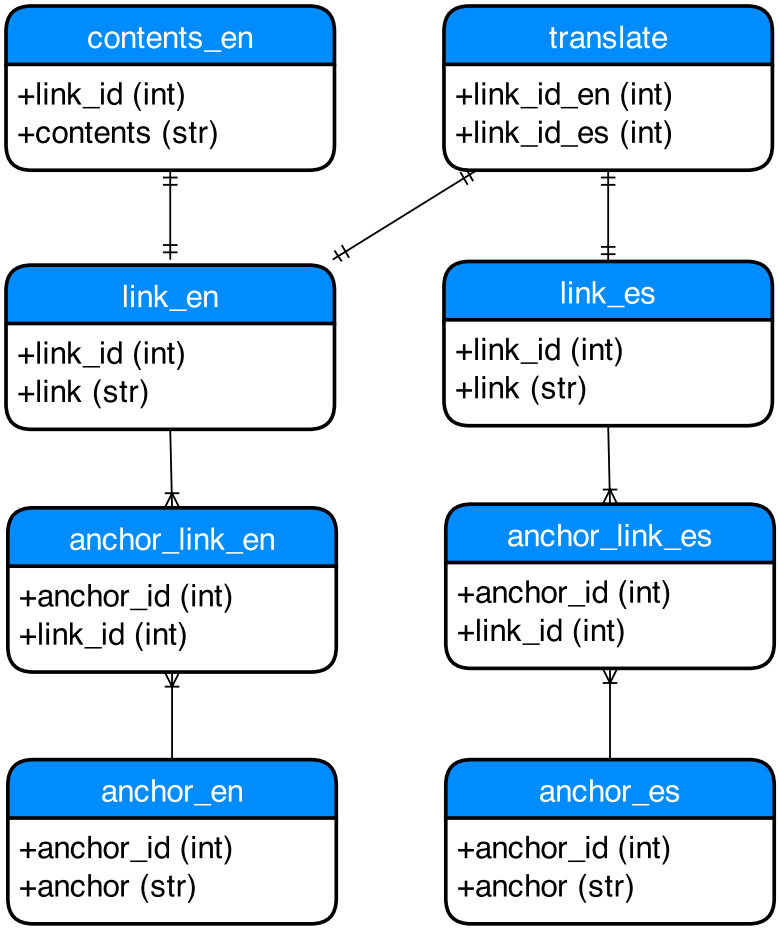


Figure 4.8: Database diagram

Chapter 5

Experiments and results

In this chapter, we will describe our set of experiments and present and analyze the results. In the first section we will describe our dataset and the trivial algorithms that we will use as a reference. The following four sections will describe each one of our proposed methods, and their configurations will be optimized using the training set. The methods are not given in any particular order as the experiments have been conducted independently from each other, with the exception of Method 3, that has been based on Method 1. Finally, in the sixth section, we will test our candidates against the test set and will be compared with the other Replab participants.

5.1 Experimental framework

5.1.1 Dataset

The dataset that we will use is the one that was constructed for the 2013 edition of the Replab competition. It includes a total of 61 entities from 4 different domains (Car companies, Banking companies, Universities and Music artists). For each entity at least 2.200 tweets where crawled during the period of time from 1 June 2012 to 31 Dec 2012. The first 700 tweets for each domain are selected as the training set and the last 1500 as the testing set so that there is a temporal separation between both sets. The corpus also includes background tweets for each entity (up to 50.000 tweets per entity) that correspond to the tweets that lie temporally between the training and testing sets. The background data will not be used in this project.

The following table shows for each domain the number of entities, total number of tweets, number of tweets on the training set, number of tweets on the testing set and number of tweets in each language (English or Spanish).

	All	Automotive	Banking	Universities	Music/Artist
Entities	61	20	11	10	20
Training No. Tweets	45.679	15.123	7.774	6.960	15.822
Test No. Tweets	96.848	31.785	16.621	14.944	33.498
Total No. Tweets	142.527	46.908	24.395	21.904	49.320
No. Tweets EN	113.544	38.614	16.305	20.342	38.283
No. Tweets ES	28.983	8.294	8.090	1.562	11.037

Table 5.1: RepLab 2013 dataset

Both the training and testing sets were labeled by a group of thirteen experts for the purpose of training and evaluating algorithms that will perform the tasks described in the replab project. In our work we will focus on the clustering subtask.

The next table shows the number of topics per set and the average number of tweets per topic, which is 17.77 for the whole dataset but goes from 14.40 in the training set to 21.14 in the test set. The training set contains 3.813 different topics, the test set 5.757 different topics, for a total of 9.570 different topics in the whole dataset.

	All	Automotive	Banking	Universities	Music/Artist
Training No. Topics	3.813	1.389	831	503	1.090
Training Average No. Tweets per Topic	14.40	12.36	11.35	17.57	16.53
Test No. Topics	5.757	1.959	1.121	1.035	1.642
Test Average No. Tweets per Topic	21.14	18.42	18.95	21.78	24.74
Total No. Topics	9.570	3.348	1.952	1.538	2.732
Total Average No. Tweets per Topic	17.77	15.39	15.15	19.67	20.64

Table 5.2: RepLab 2013 dataset for the Topic Detection Task

5.1.2 Trivial algorithms

We will define two trivial algorithms that we will use as a reference point. They consist in simply creating a cluster for each element (allinall), and creating a single cluster that will contain all elements in it (allinone).

5.1.3 Performance evaluation

For each of the proposed approaches, we will optimize the parameters with the training set. For that, we need to define an evaluation metric to be able to choose the best performing configuration for every set of runs. We propose to use a combination of the two measures that have been used to evaluate the Replab 2013 algorithms [24]. These are the B-cubed F-Measure, and the UIR measure. The UIR measure compares two algorithms. In order to obtain an absolute performance metric for a single algorithm we will use the methodology used in [24], that is, we will count how many times the algorithm has an UIR greater

or equal to 0.2 with the rest of algorithms. We will refer to this value as UIR$\geq$ 0.2. We will combine both measures (B-cubed F-Measure and UIR$\geq$ 0.2) by taking the average of the normalized versions of such measures. The normalized versions are calculated by dividing the absolute value by the maximum value obtained in the set of runs. We will refer to the combined measure as UIR+F, and it will always be our criteria for optimizing the configuration of our system on the training set.

In this chapter we will show the results of the different runs using tables with the same format. The first column will specify the rank. The second and third columns will show the name of the algorithm and its UIR$\geq$ 0.2 score in decreasing order. The fourth and fifth columns will show the name of the algorithm and its B-cubed F-Measure in decreasing order. Finally, the sixth and seventh columns will show the combined measure UIR+F also in decreasing order. All the numbers will be rounded to two digits, but the order will be based on the unrounded values.

5.2 Method 1: tweet clustering

This method will cluster the tweets using a simple similarity measure between them and using the Louvain algorithm. We will use the testing set to optimize the parameters. At this point we have a system or algorithm that accepts many different configurations. Being strict, each configuration corresponds to a different algorithm with different results. Our goal in this section is to optimize the system configuration to maximize the performance on the training set. Since the number of possible configurations is too big we can't simply try all of them, so we need to employ a different strategy. Our proposed strategy is the following: (1) try running the louvain method with all possible combinations of similarity measures and weighting schemes. (2) Select the best performing configuration using Arithmetic mean (or sum) and Harmonic mean (or F-measure) and try different relative weights on them. (3) Select the best performing configuration from points 1 and 2 and test different preprocessing schemes on it. Since we can't try all possible combination of preprocessing schemes, we will try to find the optimal configuration by making changes to the preprocessing scheme emulating a hill climbing algorithm. (4) Select the best performing configuration from point 3 and test different alpha parameters. Since this configuration does not require any html processing, and the running time of the algorithms might be important for this task, we will select it to be run on the test set and we will call it *jordi_1a*. (5) Select the best performing configuration from point 4 (*jordi_1a*) and try different html enrichment methods. (6) The best configuration from point 5 will be our selected candidate to be run on the test set. We will call it *jordi_1b*. In figure 5.1 we can see a diagram of our optimization strategy.

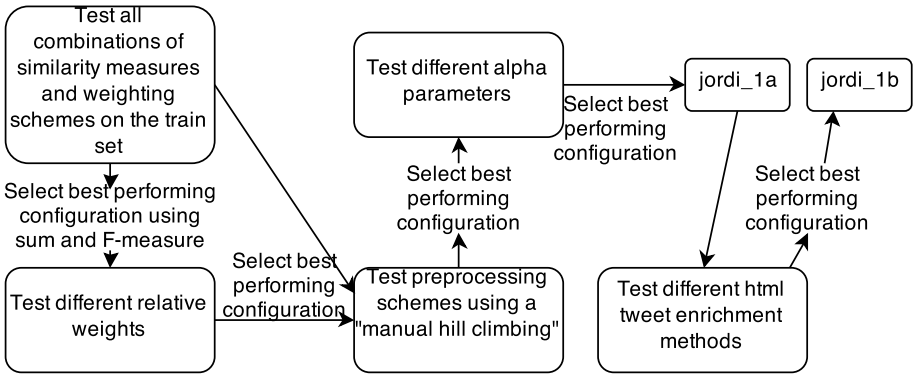


Figure 5.1: Optimization strategy for tweet clustering

5.2.1 Similarity measures and weighting schemes

In this section we will try running the louvain method with all possible combinations of similarity measures and weighting schemes.

We have three similarity measures: Euclidean, Cosine and Jaccard. And five simple weighting schemes: binary, tfidf, idf, len, loglen (len is the lenght of the term, and loglen is the logarithm of the length). And four ways of combining two simple weighting schemes: arithmetic mean or sum, harmonic mean, product and geometric mean. Both the arithmetic mean and the harmonic mean allow to set relative weights to each of the components, that is, give more importance to one component that the other. For instance, if we wanted to combine term length and tfidf but giving more importance to the later, we could use a weighted sum with a weight of 20% for the first element and 80% for the second. Similarly we can use different β values in the harmonic mean to give different relative weight to the elements.

The Geometric mean is defined as the nth root of the product of its elements:

$$\left(\prod_{i=1}^{n} a_i\right)^{1/n} = \sqrt[n]{a_1 a_2 \cdots a_n} \tag{5.1}$$

We will try each simple weighting scheme, and every combination of tfidf or idf with len or loglen using all four methods of combining them. It gives a total of 21 weighting schemes that we summarize in table 5.3 (we denote the geometrical and harmonic means with G and H).

len	len × idf	len + idf	len G idf	len H idf
loglen	len × tfidf	len + tfidf	len G tfidf	len H tfidf
idf	loglen × idf	loglen + idf	loglen G idf	loglen H idf
tfidf	loglen × tfidf	loglen + tfidf	loglen G tfidf	loglen H tfidf
binary				

Table 5.3: List of weighting schemes

We have a total of 21 weighting schemes and 3 similarity measures that gives us a total of 63 configurations to run on the training set. In tables 5.4 and 5.5 we show the results with the evaluation metrics described above (UIR$\geq$ 0.2, B-Cubed F-Measure and UIR+F).

Rank	Algorithm	UIR$\geq$ 0.2	Algorithm	F-Measure	Algorithm	UIR+F
1	euc,len × tfidf	59	euc,len × idf	0.28	euc,len × tfidf	0.99
2	euc,len × idf	57	euc,len × tfidf	0.28	euc,len × idf	0.98
3	euc,0.5×len + tfidf	56	euc,loglen × tfidf	0.27	euc,loglen × idf	0.93
4	euc,loglen × idf	53	euc,loglen × idf	0.27	euc,0.5×len + tfidf	0.93
5	euc,len F_2 tfidf	52	cos,len × idf	0.27	euc,loglen × tfidf	0.90
6	euc,idf	50	cos,len × tfidf	0.27	euc,len F_2 tfidf	0.90
7	euc,loglen × tfidf	49	cos,loglen × tfidf	0.26	euc,idf	0.88
8	euc,loglen + idf	49	cos,loglen × idf	0.26	euc,loglen + idf	0.87
9	euc,len F idf	45	euc,len F idf	0.26	euc,len F idf	0.84
10	euc,len G idf	43	euc,idf	0.26	euc,len G idf	0.81
11	euc,len G tfidf	41	euc,len F_2 tfidf	0.26	euc,len F tfidf	0.79
12	euc,loglen + tfidf	41	euc,0.5×len + tfidf	0.26	euc,len G tfidf	0.79
13	euc,tfidf	40	euc,len F tfidf	0.26	euc,loglen + tfidf	0.79
14	euc,len F tfidf	40	euc,loglen + idf	0.25	euc,tfidf	0.78
15	euc,loglen G idf	38	euc,len G idf	0.25	euc,loglen G idf	0.76
16	jac,len × idf	37	euc,len G tfidf	0.25	jac,len × idf	0.75
17	euc,len + idf	36	euc,loglen + tfidf	0.25	euc,len + idf	0.73
18	euc,len + tfidf	35	euc,tfidf	0.25	euc,len + tfidf	0.72
19	euc,loglen G tfidf	33	euc,loglen G idf	0.25	euc,loglen G tfidf	0.71
20	cos,idf	30	jac,len × idf	0.25	cos,loglen × tfidf	0.68
21	jac,loglen × idf	27	allinall	0.24	cos,idf	0.68
22	cos,loglen × tfidf	26	euc,loglen G tfidf	0.24	jac,loglen × idf	0.66
23	cos,tfidf	25	jac,loglen × idf	0.24	cos,tfidf	0.63
24	cos,len F idf	24	euc,len + idf	0.24	cos,len F idf	0.63
25	cos,loglen + tfidf	23	cos,idf	0.24	cos,loglen + idf	0.61
26	cos,loglen + idf	23	cos,len F idf	0.24	cos,loglen + tfidf	0.61
27	jac,len × tfidf	22	euc,len + tfidf	0.24	cos,len × tfidf	0.61
28	cos,len + idf	21	euc,loglen F idf	0.24	jac,len × tfidf	0.61
29	euc,loglen F tfidf	21	euc,2×len + tfidf	0.24	euc,loglen F tfidf	0.59
30	cos,len G tfidf	20	cos,len G idf	0.24	cos,len G idf	0.59
31	cos,len G idf	20	cos,len F tfidf	0.24	cos,len G tfidf	0.59
32	cos,len F tfidf	19	cos,loglen + idf	0.24	euc,loglen F idf	0.58
33	euc,loglen F idf	19	jac,len × tfidf	0.24	cos,len F tfidf	0.58
34	cos,len + tfidf	17	cos,tfidf	0.24	cos,loglen × idf	0.58
35	cos,len × tfidf	16	cos,loglen + tfidf	0.23	cos,len + idf	0.58
36	cos,loglen G tfidf	16	cos,len G tfidf	0.23	euc,2×len + tfidf	0.56
37	cos,loglen G idf	16	euc,loglen F tfidf	0.23	cos,len × idf	0.56
38	euc,2×len + tfidf	16	euc,len $F_{0.5}$ tfidf	0.23	cos,loglen G idf	0.55
39	jac,loglen × tfidf	15	jac,loglen × tfidf	0.23	cos,loglen G tfidf	0.54
40	jac,idf	14	cos,loglen G idf	0.23	jac,loglen × tfidf	0.54

Table 5.4: Results for training set for all combinations of similarities and weighting schemes

Rank	Algorithm	UIR$\geq$ 0.2	Algorithm	F-Measure	Algorithm	UIR+F
41	cos,loglen $\times$ idf	14	cos,loglen G tfidf	0.23	cos,len + tfidf	0.54
42	euc,len	12	cos,len + idf	0.23	jac,idf	0.52
43	jac,loglen + idf	11	jac,idf	0.23	euc,len	0.49
44	cos,len $\times$ idf	10	cos,loglen F idf	0.22	jac,loglen + idf	0.48
45	jac,len G idf	7	cos,len + tfidf	0.22	jac,len G idf	0.45
46	jac,len + idf	7	cos,loglen F tfidf	0.22	euc,len $F_{0.5}$ tfidf	0.44
47	jac,tfidf	5	jac,len F idf	0.22	allinall	0.44
48	jac,len F idf	5	jac,len G idf	0.22	jac,len + idf	0.43
49	jac,loglen + tfidf	5	euc,len	0.22	jac,len F idf	0.43
50	jac,loglen G idf	5	jac,loglen + idf	0.22	jac,tfidf	0.42
51	cos,len	5	jac,len F tfidf	0.21	cos,loglen F idf	0.42
52	euc,loglen	5	jac,tfidf	0.21	euc,loglen	0.42
53	jac,len + tfidf	4	jac,len G tfidf	0.21	jac,loglen G idf	0.42
54	jac,loglen F idf	4	jac,loglen F idf	0.21	cos,loglen F tfidf	0.42
55	jac,len	3	euc,loglen	0.21	jac,loglen F idf	0.41
56	jac,len G tfidf	3	jac,loglen G idf	0.21	jac,loglen + tfidf	0.41
57	jac,len F tfidf	3	jac,len + idf	0.21	cos,len	0.41
58	jac,loglen	3	jac,loglen + tfidf	0.21	jac,len F tfidf	0.41
59	jac,loglen G tfidf	3	jac,loglen G tfidf	0.21	jac,len G tfidf	0.40
60	jac,loglen F tfidf	3	jac,loglen F tfidf	0.21	jac,loglen G tfidf	0.39
61	cos,loglen	3	cos,len	0.21	jac,loglen F tfidf	0.39
62	cos,loglen F tfidf	3	jac,len	0.20	jac,len	0.38
63	cos,loglen F idf	3	jac,loglen	0.20	jac,loglen	0.38
64	euc,len $F_{0.5}$ tfidf	3	cos,loglen	0.20	cos,loglen	0.37
65	jac,1	1	jac,len + tfidf	0.19	jac,len + tfidf	0.37
66	euc,1	1	euc,1	0.19	euc,1	0.34
67	cos,1	0	jac,1	0.18	jac,1	0.33
68	allinone	0	cos,1	0.17	cos,1	0.31
69	allinall	0	allinone	0.12	allinone	0.22

Table 5.5: Results for training set for all combinations of similarities and weighting schemes (continuation)

Relative weight

In order to find if we can improve the performance of the addition and F1 methods of combining weights, we are going to select the best performing run using each one of this methods and we are going to change the relative weights of their components. For the addition we will try multiplying the first term by 2 and 0.5. For the harmonic mean we will try different values of β (0.5 and 2). Which gives us an extra 4 runs. The results are included in the tables 5.4 and 5.5. We denote the value of β as a subindex of F or in the algorithm name in the table.

We have also selected the best performing runs using methods of combining weights that allow to give a relative weight and rerun them using different

relative weight values. For the sum method, the best performing one using the UIR+F column as the criteria is Euclidean using len + $tfidf$ as weights. So we try two new runs this time multiplying length by 2 and by 0.5. We get an important improvement when we multiply by 0.5, placing the new run from 17th position to 4th position, whereas when we multiply length by 2 we drop to position 36. This suggests that tfidf should be weighted more than length when we combine both measures. We observe the same phenomenon when we change the relative weight through β using the harmonic mean. This time when we increase the relative weight for tfidf we go from position 11th to 6th, and we drop to position 46th when we increase the relative weight of length instead.

Results

The first thing that we observe in the results is that the Euclidean measure seems to consistently perform better that the other two similarities, while the Jaccard similarity seems to be the worse performing one of the three. Another observation is that the weighting scheme $len \times tfidf$ consistently outperforms $tfidf$. Also, taking the logarithm of the lenght seems to perform worse than by just using the lenght. According to the UIR+F measure (seventh column), from the total of 67 runs, the best performing algorithm is: *Louvain with euclidean similarity and len* $\times$ $tfidf$ *weighting scheme*. This algorithm ranks second for the B-Cubed F-Measure and first for the UIR, and it will be the selected one for further improvements.

5.2.2 Tweet Preprocessing

At this point we will try to improve the results obtained by the best performing run of the previous experiment (*Louvain with euclidean similarity and len*$\times tfidf$ *weighting scheme*). We have the following set of 13 preprocessing elements that we can combine.

1. **lower**: convert uppercase to lowercase.
2. **toascii**: eliminate accents and diacritics. We use the unidecode [69] python package to convert from unicode to ascii.
3. **alpha only**: convert all characters different from a-z to spaces or separators.
4. **delete small terms**: remove terms with small lengths.
5. **delete links**: remove http links.
6. **duplicate/triplicate hashtags**: used to increase the weight of the hashtags.
7. **duplicate/triplicate user references**: used to increase the weight of references to users.

8. **stem**: used to get the root form of the terms so that different lexical forms of the same lemma can be found using simple string comparison after stemming. Two different algorithms are used. porter and porter2.

9. **stopwords**: deletion of common terms. Two lists of stop terms are used. One taken from mysql [66] and the other from oracle [67].

The total number of possible combinations is $2^{13} = 8192$. Since the computation time of every run is relatively long, it is not possible for us to run every possible configuration, so we will use another approach. We will emulate a hill climbing algorithm. The initial state will be no preprocessing at all. We will try every preprocessing element individually, and we will choose the best performing one for the next iteration. In the second iteration we will select every preprocessing element in combination with the one selected in the first iteration and we will choose the best performing configuration found for the next iteration. The process goes on until no further improvements can be found.

In table 5.6 we describe the label that we have given to each configuration that we have tried. In table 5.7 we show the results of every configuration.

label	lower	toascii	alpha	mysql	oracle	rm1	rm2	rm3	rmlink	porter	porter2	#×2	#×3	@×2	@×3
#×2,@×2	x	x	x									x		x	
#×2,@×2,rm1	x	x	x			x						x		x	
#×3,@×2	x	x	x										x	x	
#×2,@×3	x	x	x									x			x
#×2	x	x	x									x			
@×2	x	x	x											x	
rm1	x	x	x			x									
alpha	x	x	x												
@×3	x	x	x												x
#×3	x	x	x										x		
rmlink	x	x	x						x						
porter2	x	x	x								x				
rm2	x	x	x				x								
porter	x	x	x							x					
rm3	x	x	x					x							
toascii	x	x													
oracle	x	x	x		x										
lower	x														
mysql	x	x	x	x											
none															

Table 5.6: Preprocessing schemes configurations

Rank	Algorithm	UIR≥ 0.2	Algorithm	F-Measure	Algorithm	UIR+F
1	#×2,@×2	9	#×2,@×2,rm1	0.30	#×2,@×2	1.00
2	#×2,@×2,rm1	8	#×2,@×2	0.30	#×2,@×2,rm1	0.94
3	alpha	7	#×3,@×2	0.30	#×3,@×2	0.89
4	#×2,@×3	7	#×2,@×3	0.30	#×2,@×3	0.89
5	#×3,@×2	7	#×2	0.30	#×2	0.89
6	#×2	7	#×3	0.30	@×2	0.88
7	@×2	7	@×2	0.30	rm1	0.88
8	@×3	7	rm1	0.30	alpha	0.88
9	rm1	7	rm2	0.30	@×3	0.88
10	#×3	6	rmlink	0.29	#×3	0.83
11	rmlink	6	alpha	0.29	rmlink	0.83
12	porter2	4	@×3	0.29	porter2	0.71
13	porter	3	rm3	0.29	rm2	0.66
14	rm2	3	porter	0.29	porter	0.65
15	oracle	0	porter2	0.29	rm3	0.49
16	lower	0	toascii	0.29	toascii	0.48
17	toascii	0	oracle	0.29	oracle	0.48
18	rm3	0	lower	0.29	lower	0.48
19	mysql	0	mysql	0.28	mysql	0.47
20	none	0	none	0.28	none	0.47

Table 5.7: Preprocessing schemes results

After experimenting with different configurations with the process described below (manual hill climbing) we discover that the best configuration for preprocessing is #×2,@×2. That is, we convert to lower case, to ascii, keep only alpha characters, and finally we duplicate both the hashtags and the user references.

It is interesting to note that the configuration without preprocessing (*none*) takes the last position in the ranking, showing that all the proposed preprocessing methods are beneficial when used individually. The optimal configuration found does not include all of them is because some combinations of preprocessing methods do not work well together.

5.2.3 Selecting the modularity granularity

As we have seen in the previous chapter, we can select the granularity of the communities by changing the α parameter in the extended modularity definition. So far we have always been using $\alpha = 1$. In this set of runs we will try different values of α and we will evaluate each one of them. The values that we will try are the numbers from 1 to 10, 15, 20, 50 and 100. The *base* algorithm is the best performing one in the previous section: *Louvan with Euclidean similarity, len × tfidf weighting scheme and #×2,@×2 preprocessing.*

Rank	α	UIR$\geq$ 0.2	α	F-Measure	α	UIR+F
1	1	0	7	0.32	7	1.00
2	2	0	6	0.32	6	1.00
3	3	0	5	0.32	5	0.99
4	4	0	8	0.32	8	0.99
5	5	0	9	0.32	9	0.99
6	6	0	4	0.32	4	0.99
7	7	0	3	0.32	3	0.98
8	8	0	15	0.32	15	0.97
9	9	0	2	0.31	2	0.97
10	15	0	20	0.31	20	0.95
11	20	0	1	0.30	1	0.93
12	50	0	50	0.29	50	0.89
13	100	0	100	0.27	100	0.83

Table 5.8: Comparison of different values of α

If we look at the UIR column, we see that all values are 0, probably because even though the α parameters are different, the algorithms are not different enough to make a big difference, so we never obtain a UIR greater or equal to 0.2. On the other hand we can see important differences in the B-Cubed F-Measures, that will allow us to select the optimum α value. In table 5.8 we can see at the top the configurations $\alpha = 6$ and $\alpha = 7$ both with the same value for $UIR + F$. Since we need to select one configuration only, we will select the one at the top $\alpha = 7$, since the order is given using the unrounded values and we do not have any reason to choose the configuration in the second position instead. This configuration (*Louvain with Euclidean similarity with len $\times$ tfidf as weighting scheme and #$\times$2,@$\times$2 as preprocessing and* $\alpha = 7$) will be selected to be run on the test set, and we will call it *jordi_1a*.

5.2.4 Html tweet enrichment

In this section we will run experiments that follow the html links for the tweets that have them. We will extract relevant information from the html and append it to the tweet. One strategy is to extract the title of the page, as it usually contains a very short summary of the contents of the webpage. Another strategy is to find the keywords of the document. We use again tfidf to measure the weights of each term for every website. In order to find the idf factor, we create a corpus of documents containing all websites that are linked in our set of tweets. The html is parsed using the nltk [70] python package. Once we have the weight of every term in the html document, we can extract the top n terms with higher weights and append them to the tweet. This will enrich the information contained in the tweet and probably benefit the clustering algorithm.

We run a combination of the two strategies described above. We choose as the *jordi_1a* algorithm the best performing one in the previous section: *Louvain with Euclidean similarity with len $\times$ tfidf as weighting scheme and #$\times$2,@$\times$2*

as preprocessing and $\alpha = 7$, and we run over it every combination of adding or not the html title, and appending the top 5, 10, 15, 20 and 25 most relevant (by tfidf) terms from the html document. We show the results in the following table:

Rank	Algorithm	UIR$\geq$ 0.2	Algorithm	F-Measure	Algorithm	UIR+F
1	jordi_1a	1	+title,+top15	0.33	+title,+top15	1.00
2	+title	1	+title,+top25	0.33	+title,+top10	1.00
3	+title,+top5	1	+title,+top10	0.33	+title,+top5	1.00
4	+title,+top10	1	+title,+top5	0.33	jordi_1a	1.00
5	+title,+top15	1	+title,+top20	0.32	+title	1.00
6	+title,+top20	0	+top15	0.32	+title,+top25	0.50
7	+title,+top25	0	jordi_1a	0.32	+title,+top20	0.50
8	+top5	0	+title	0.32	+top15	0.50
9	+top10	0	+top5	0.32	+top5	0.49
10	+top15	0	+top20	0.32	+top20	0.49
11	+top20	0	+top10	0.32	+top10	0.49
12	+top25	0	+top25	0.32	+top25	0.49

Table 5.9: Html tweet enrichment results

In table 5.9 we can see several configurations at the top of the table with the same score for $UIR + F$. As we did in the previous section, we will select the configuration that appears at the first position. The reason is that even thought the scores have been rounded, the list is given sorted by the unrounded values. We are aware that some of the algorithms take more time and space than others, but our criteria for selection is based on clustering performance only. Therefore we select the configuration *Louvain with Euclidean similarity with len × tfidf as weighting scheme, #×2,@×2 as preprocessing,* $\alpha = 7$ *and +title,+top15 html enrichment* as our candidate to be run on the test set and we will call it *jordi_1b*.

5.3 Method 2: html clustering

The challenge of clustering tweets is caused by the short size of the documents that are limited to 140 characters. In this set of experiments, what we are going to do is to cluster the html documents instead of the tweets referencing them. The problem is that not all tweets provide links to websites. We propose to divide the tweets into two groups, the ones containing links to html documents and the ones without, so that we can cluster both groups independently. In one of them we will use the tweets as the documents, and in the other we will use the html as the documents. Our hypothesis is that since we will get the benefit of clustering big html documents, the overall clustering will be more accurate. We will use two clustering algorithms for each group: the *louvain using cosine similarity with tfidf* and the *allinall*. This gives us a total of 4

algorithms. The cosine similarity with tfidf is chosen for being a standard in the clustering literature. With this experiments we will see what is the overall result of clustering only the html group, clustering only the tweet group, clustering both the html and the tweet groups and not clustering anything. After selecting the best strategy, we will try to improve it by increasing the weight of the terms appearing in the title of the html document, and by changing the α parameter.

Name	Tweets group algorithm	Html group algorithm
html/tweet	louvain	louvain
html	allinall	louvain
tweet	louvain	allinall
allinall	allinall	allinall

Table 5.10: Html clustering algorithms

The results are shown in the following table:

Rank	Algorithm	UIR$\geq$ 0.2	Algorithm	F-Measure	Algorithm	UIR+F
1	tweet/html	0	html	0.36	html	1.00
2	tweet	0	tweet	0.36	tweet	1.00
3	html	0	allinall	0.24	allinall	0.67
4	allinall	0	tweet/html	0.22	tweet/html	0.59

Table 5.11: Html clustering results

Here we can see that the best performing strategy is to only cluster the html group leaving the tweet group unclustered. As in previous sections, we will consider that the best algorithm is the one at the top even if the scores are the same. This is a result that we did not expect, and we will analyze in the next section: the uncluster hack.

We would like to try two more experiments based on the best performing configuration from the set of runs above (html clustering only). First, we want to give more weight to the title of the html page, since it will probably contain relevant terms. We accomplish that by appending the title ten times at the end of the parsed html document. We are effectively increasing by ten the tf factor in the tfidf weight of each term appearing in the title. We are going to pick the best performing configuration of the two, and change its α parameter to 5. The following table shows the results.

Rank	Algorithm	UIR$\geq$ 0.2	Algorithm	F-Measure	Algorithm	UIR+F
1	html+title$\times$10	1	html	0.36	html+title$\times$10	1.00
2	html	0	html+title$\times$10	0.36	html	0.50
3	html+title$\times$10,α=5	0	html+title$\times$10,α=5	0.28	html+title$\times$10,α=5	0.39

Table 5.12: Html clustering results (2)

As we can see, increasing the weight of the title of the html document gives us better results (based on $UIR+F$), whereas changing the parameter α does not. We then select the louvain algorithm that clusters only the tweets containing html links and leaves the rest in their own cluster. The similarity measure is the cosine of the html document with the title appended ten times, using a tfidf as the weighting scheme. We will call it *jordi_2*.

5.4 Method 3: uncluster hack

The results from the experiments run in the previous section seemed a bit counter intuitive The problem that we find is the following: if we cluster only the tweet group we perform better than allinall, if we cluster only the html group we perform better than allinall, but if we cluster both at the same time we perform worse than the allinall. This seems to break a constraint that we propose and that we will describe in this section. We call it, the double improvement consistency.

5.4.1 Double improvement consistency

Let S be a set of items belonging to categories $L_1 ... L_n$. Let $S1$ and $S2$ be disjoint subsets of S such as that $S1 \cup S2 = S$. Let D_1, D_2 be two cluster distributions of $S1$, and D_3, D_4 two cluster distributions of $S2$. Then the following expression should hold true.

$$Q(S, D_1|D_3) < Q(S, D_1|D_4) \wedge Q(S, D_1|D_3) < Q(S, D_2|D_3) \implies Q(S, D_1|D_3) < Q(S, D_2|D_4)$$

Double improvement consistency with the B-Cubed F-Measure

In this section we will prove that the B-Cubed F-Measure does not fullfill the double improvement consistency. We do it by simply giving a counter example since that is sufficient. We have illustrated the example in figure 5.2. In this case, and following the nomenclature of the double improvement consistency definition, S are eight balls with categories encoded as colors (two white, three black and three gray). The set is split into two groups of four balls each. In the figure we have arranged both groups in the left hand side and right hand side. D_1 and D_3 are clustering distributions that cluster each element in its own cluster, while the D_2 and D_4 clustering distributions cluster all elements into one big cluster. We just need to calculate the B-Cubed F-Measure of each example to show that the double improvement consistency does not hold.

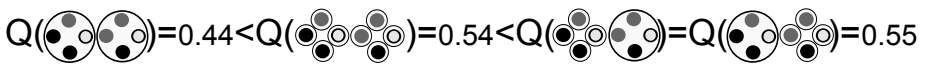


Figure 5.2: Double improvement inconsistency for B-cubed F-Measure

5.4.2 The uncluster hack

There is another surprising result that we can see from table 5.11. That is the high B-Cubed F-Measure obtained from the *tweet* run. This run is essentially a *louvain clustering with cosine and tfidf* using tweet data with the particularity that we are only clustering tweets without html links, leaving all other tweets unclustered in their own cluster. Nevertheless, this run gets a B-Cubed F-Measure of 0.36 (figure 5.11) that is much higher than the 0.24 obtained when we cluster all tweets (table 5.4), regardless of whether they contain links to html documents or not. This result is very surprising, and makes us believe that there might be a problem with the F-Measure. Our hypothesis is that randomly leaving a number of documents unclustered might yield to a better B-Cubed F-Measure. So we are going to design the following experiment to prove our hypothesis: we are going to include a new preprocessing element that will uncluster a number of randomly selected nodes. We can accomplish it by simply setting documents to the empty document, as they will be automatically unclustered (clustered in their own cluster) by any clustering algorithm. Every document will be emptied with probability *rm* during the preprocessing phase, and we are going to try several different *rm* parameters. In the following table we will try different values of *rm* using the *jordi_1* algorithm. We will try the following values for *rm*: 0, 0.5, 0.6, 0.65, 0.7, 0.75, 0.8 and 0.9.

Rank	Algorithm	UIR$\geq$ 0.2	Algorithm	F-Measure	Algorithm	UIR+F
1	jordi_1	0	jordi_1,rm.75	0.40	jordi_1,rm.75	1.00
2	jordi_1,rm.7	0	jordi_1,rm.7	0.40	jordi_1,rm.7	1.00
3	jordi_1,rm.75	0	jordi_1,rm.8	0.39	jordi_1,rm.8	0.99
4	jordi_1,rm.65	0	jordi_1,rm.65	0.39	jordi_1,rm.65	0.99
5	jordi_1,rm.8	0	jordi_1,rm.6	0.38	jordi_1,rm.6	0.96
6	jordi_1,rm.9	0	jordi_1,rm.5	0.36	jordi_1,rm.5	0.92
7	jordi_1,rm.6	0	jordi_1,rm.9	0.36	jordi_1,rm.9	0.91
8	jordi_1,rm.5	0	jordi_1	0.30	jordi_1	0.76

Table 5.13: Preprocessing schemes results

If we consider the B-Cubed F-Measure only, we can see clearly that the hypothesis turns out to be true, reaching a maximum value of 0.40 for *rm*=0.75 (for *rm*=0.7 is smaller even though it can't be shown in the table). We will select this configuration for the test set and we will call it *jordi_3*.

The results of these experiments lead us to believe that the B-Cubed F-Measure might not be the best metric that we can use for this task. Randomly deleting the text from 75% of the tweets cannot be a good strategy, yet the B-Cubed F-Measure indicates otherwise. This fact in addition to the double improvement inconsistency might be good indicators that another metric should be used for this task in particular.

5.5 Method 4: Fuzzy Weighted Jaccard on wikified tweets

In this section we will experiment using the louvain method with the Fuzzy Weighted Jaccard similarity measure. As we have seen before, the Fuzzy Weighted Jaccard is a similarity measure between sentences or tweets that needs another similarity measure across the terms of the sentences. For this reason we use the wikified tweets, that assign a Wikipedia article to each term. Now we can calculate the similarity of two terms as the similarity of their corresponding Wikipedia articles.

The Fuzzy Weighted Jaccard accepts a parameter f that controls the fuzziness. We will set that value to 0 and we will optimize the value of α against the training set. We will try the values of α increasingly until we cannot improve the results anymore. Once we find the value that yields with the best results, we will set the value of α to the best found, and we will try to optimize the parameter f trying increasing values until we cannot improve the results anymore. We show the results in table 5.14.

Rank	Algorithm	UIR$\geq$ 0.2	Algorithm	F-Measure	Algorithm	UIR+F
1	f=0.25,α=7	1	f=0,α=7	0.32	f=0.25,α=7	1.00
2	f=0,α=7	0	f=0,α=8	0.31	f=0,α=7	0.50
3	f=0,α=3	0	f=0.5,α=7	0.31	f=0,α=8	0.50
4	f=0,α=4	0	f=0.25,α=7	0.31	f=0.5,α=7	0.50
5	f=0,α=5	0	f=0,α=6	0.31	f=0,α=6	0.50
6	f=0,α=6	0	f=0,α=7	0.31	f=0,α=7	0.50
7	f=0,α=8	0	f=0.1,α=7	0.31	f=0.1,α=7	0.50
8	f=0.5,α=7	0	f=0,α=5	0.31	f=0,α=5	0.49
9	f=0.1,α=7	0	f=0,α=4	0.30	f=0,α=4	0.48
10	f=0,α=7	0	f=0,α=3	0.28	f=0,α=3	0.45

Table 5.14: Fuzzy Jaccard results

The optimal value found for α is 7, and after setting α to its optimal value we find f=0.25 to be the optimal configuration. These experiments show us that the Fuzzy Weighted Jaccard is able to improve the results of the weighted Jaccard. We expected this improvement to be a bit better than what we got (actually the B-Cubed F-Measure decreased a bit, the improvement comes from the UIR metric), and we attribute this small improvement due to the fact that the wikfying is a very noisy process, in the sense that very often, the tweets get related to articles that have little or nothing to do with the topic of the tweet (like a movie or music album articles with titles appearing as n-grams in the tweet). We will select the best performing run (α=7 and f=0.25) as our fourth and last candidate to be run in the test set (*jordi_4*).

5.6 Test set runs

Finally we are going to run our four candidates along with the participants of replab 2013 (a total of 38 runs) in our test set, and we will give the same performance metrics as in previous sections. Our runs are:

- **jordi_1a**: Tweet clustering enriched with html (Euclidean, len×tfidf, #×2, @×2, α=7)
- **jordi_1b**: Tweet clustering (Euclidean, len×tfidf, #×2, @×2, α=7+html title + top 15 html keywords)
- **jordi_2**: Html clustering (html+title*10)
- **jordi_3**: uncluster hack (jordi_1, rm=0.75)
- **jordi_4**: Fuzzy Jaccard of wikified tweet (α=7, f=0.25)

We can see the results in table 5.15:

Rank	Algorithm	UIR$\geq$ 0.2	Algorithm	F-Measure	Algorithm	UIR+F
1	**jordi_2**	17	**jordi_3**	0.37	**jordi_2**	0.94
2	uned_orm_2	13	uned_orm_2	0.37	uned_orm_2	0.88
3	oneinone	11	**jordi_2**	0.33	**jordi_3**	0.73
4	uamclyr_7	9	**jordi_1b**	0.3	reina_2	0.64
5	reina_2	8	reina_2	0.3	uamclyr_7	0.59
6	**jordi_3**	8	uned_orm_3	0.3	oneinone	0.57
7	uamclyr_8	5	lia1	0.3	**jordi_1b**	0.53
8	**jordi_1a**	4	uned_orm_4	0.3	**jordi_1a**	0.51
9	**jordi_1b**	4	uned_orm_5	0.3	uned_orm_7	0.51
10	reina_1	4	**jordi_1a**	0.29	uned_orm_3	0.49
11	uned_orm_7	4	lia2	0.29	uned_orm_4	0.49
12	uned_orm_3	3	uned_orm_7	0.29	uned_orm_5	0.49
13	uned_orm_4	3	lia3	0.29	**jordi_4**	0.47
14	uned_orm_5	3	**jordi_4**	0.28	uned_orm_6	0.46
15	uned_orm_6	3	lia4	0.28	lia2	0.45
16	uamclyr_5	3	uned_orm_6	0.28	lia3	0.45
17	uamclyr_6	3	uamclyr_5	0.25	reina_1	0.44
18	**jordi_4**	3	uamclyr_6	0.25	lia4	0.44
19	lia2	2	uamclyr_7	0.24	uamclyr_5	0.42
20	lia3	2	reina_1	0.24	uamclyr_6	0.42
21	lia4	2	uned_orm_1	0.22	uamclyr_8	0.42
22	uamclyr_3	1	baseline	0.22	lia1	0.40
23	baseline	0	uamclyr_3	0.22	uamclyr_3	0.33
24	allinone	0	uamclyr_4	0.21	uned_orm_1	0.30
25	lia1	0	uamclyr_8	0.2	baseline	0.30
26	nlp_ir_uned_1	0	uamclyr_1	0.2	uamclyr_4	0.29
27	nlp_ir_uned_10	0	uamclyr_2	0.19	uamclyr_1	0.27
28	nlp_ir_uned_2	0	oneinone	0.18	uamclyr_2	0.26
29	nlp_ir_uned_3	0	nlp_ir_uned_1	0.16	nlp_ir_uned_1	0.22
30	nlp_ir_uned_4	0	nlp_ir_uned_2	0.16	nlp_ir_uned_2	0.21
31	nlp_ir_uned_5	0	nlp_ir_uned_3	0.15	nlp_ir_uned_3	0.21
32	nlp_ir_uned_6	0	nlp_ir_uned_4	0.15	nlp_ir_uned_4	0.21
33	nlp_ir_uned_7	0	nlp_ir_uned_5	0.15	nlp_ir_uned_5	0.20
34	nlp_ir_uned_8	0	allinone	0.12	allinone	0.17
35	nlp_ir_uned_9	0	nlp_ir_uned_8	0.12	nlp_ir_uned_8	0.16
36	uned_orm_1	0	nlp_ir_uned_9	0.12	nlp_ir_uned_9	0.16
37	uamclyr_1	0	nlp_ir_uned_10	0.12	nlp_ir_uned_10	0.16
38	uamclyr_2	0	nlp_ir_uned_6	0.12	nlp_ir_uned_6	0.16
39	uamclyr_4	0	nlp_ir_uned_7	0.12	nlp_ir_uned_7	0.16

Table 5.15: Test set results

If we look at the UIR+F column, we will see that the best performing algorithm is the html clustering. In third position we find the uncluster hack.

The seventh and eight positions are occupied by the enriched tweet clustering and the simple tweet clustering, and finally the worse performing algorithm of our submits is the Fuzzy Weighted Jaccard with Wikipedia occupying the 13th position. In the F-Measure column we can see that 3 of the top 4 positions are our submissions. The top one, not surprisingly is the uncluster hack. The UIR$\geq$ 0.2 column is again lead by the html clustering. Interestingly we can find the trivial algorithm of oneinone in the third position, which seems counter intuitive as it had a relatively low B-Cubed F-Measure.

We are a bit surprised by the low rank of the Fuzzy Weighted Jaccard with Wikipedia, as it is the most elaborate algorithm. We attribute this low rank due to the noise that the wikifying step produces (linking the tweet with irrelevant pages), and with the fact that, as we proved before, the Jaccard measure performs worse than cosine, and Euclidean, the latter being the preferred one for this task.

Chapter 6

Conclusions and future work

The final chapter presents the conclusions with the main findings of our study and their implications, along with possible future lines of research in this field.

6.1 Conclusions

This study explores different methodologies for the task of tweet clustering. We revealed two problems related to the currently-accepted B-cubed F-Measure. We have demonstrated how it fails to comply with the reasonably-expected double improvement inconsistency and uncovered with the uncluster hack how it benefits excessively from leaving random elements unclustered, neither of which are desirable properties. As it is heretofore unreported, this finding opens the door for new lines of research treating these issues. A crucial step for creating a system, after all, is ensuring its proper evaluation. Furthermore, we have offered novel concepts in the field of clustering. We have introduced the extended modularity for the task of clustering and shown how it can improve the performance of the algorithms by changing the value of α. We have also proposed a new definition based on the Jaccard similarity that is able to compare documents at the semantic level. Besides proving that these two concepts can be useful for the task of tweet clustering, we believe that they can be successfully used in other disciplines, as well, since they can be tools for the clustering and comparison of objects other than documents. In our experiments, we have shown that in contradiction to current literature, the Euclidean distance performs better for this particular task than the other commonly used similarity measures, the cosine and the Jaccard index. We have further demonstrated that the tfidf weighting scheme can be improved by being multiplied by the term length. Additionally, we have shown that only a few number of preprocessing elements become beneficial for the tweet clustering task. Using the combined evaluating measure as a reference, the best algorithm that we have found–it outperforms all competitors

of Replab 2013–is to cluster the html documents referenced by the tweets and leave the other tweets unclustered.

6.2 Future work

In addition to the evidence this thesis provides for its claims, statistical hypothesis tests can be used to detect the significance of our results. In light of the aforementioned failings of the B-Cubed F-Measure, new evaluation measures would be assets to the problem of topic detection. Other possibilities include combining the Fuzzy Weighted Jaccard measure with other term similarity methods; using other resources instead of Wikipedia, such as wordnet; and using Google searches to calculate term similarities [72]. Alternatively, one may use the term similarity measures defined by the term clustering proposal of the ORM UNED submission [21]. Another area of research would entail finding the optimal values and parameters of the algorithms on a per entity basis, which we assume would allow the algorithms to better adapt to different entities. A further refinement could be the use of metadata for the clustering task; for example, the user network of followers could be used to gather extra information about the tweets. Intuitively, two users are more likely to be posting about the same topic if one is following the other. Our tests, not reported in this thesis as they are not directly related to the concrete task at hand, show that the gold standard's categories frequency distribution follows a power law. This is an interesting property that might be caused by a preferential attachment process (it is more likely that one would talk about a popular topic because it is popular, the result being a subsequent increase in the topic's popularity). Knowing this, it might be possible to design an algorithm that takes advantage of this fact by forcing the solution to follow a power law distribution as the gold standard does.

Bibliography

[1] Allan, J., Topic detection and tracking, J. Allan, Editor 2002, Kluwer Academic Publishers. p. 1-16.

[2] Mori, Masaki, Takao Miura, and Isamu Shioya. "Topic detection and tracking for news web pages." Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence. IEEE Computer Society, 2006.

[3] Brooks, Christopher H., and Nancy Montanez. "Improved annotation of the blogosphere via autotagging and hierarchical clustering." Proceedings of the 15th international conference on World Wide Web. ACM, 2006.

[4] O'Connor, Brendan, Michel Krieger, and David Ahn. "TweetMotif: Exploratory Search and Topic Summarization for Twitter." ICWSM. 2010.

[5] http://tweetmotif.com

[6] Sankaranarayanan, J., et al. TwitterStand: news in tweets. in Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM.

[7] http://twitterstand.umiacs.umd.edu/news/

[8] Shah, Devavrat, and Tauhid Zaman. "Community detection in networks: The leader-follower algorithm." arXiv preprint arXiv:1011.0774 (2010).

[9] Rosa, Kevin Dela, et al. "Topical clustering of tweets." Proceedings of the ACM SIGIR: SWSM (2011).

[10] Carter, Simon, Manos Tsagkias, and Wouter Weerkamp. "Twitter hashtags: Joint Translation and Clustering." Proceedings of the ACM WebSci'11 (2011): 1-3.

[11] Bernstein, Michael S., et al. "Eddi: interactive topic-based browsing of social status streams." Proceedings of the 23nd annual ACM symposium on User interface software and technology. ACM, 2010.

[12] Sriram, Bharath, et al. "Short text classification in twitter to improve information filtering." Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval. ACM, 2010.

[13] Rangrej, Aniket, Sayali Kulkarni, and Ashish V. Tendulkar. "Comparative study of clustering techniques for short text documents." Proceedings of the 20th international conference companion on World wide web. ACM, 2011.

[14] Jin, Ou, et al. "Transferring topical knowledge from auxiliary long texts for short text clustering." Proceedings of the 20th ACM international conference on Information and knowledge management. ACM, 2011.

[15] Karandikar, Anand. Clustering short status messages: A topic model based approach. Diss. University of Maryland, 2010.

[16] Miller, Zachary, et al. "Twitter spammer detection using data stream clustering." Information Sciences 260 (2014): 64-73.

[17] Petrović, S., M. Osborne, and V. Lavrenko. Streaming first story detection with application to Twitter. in Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics. Association for Computational Linguistics.

[18] Petrović, S., M. Osborne, and V. Lavrenko. The Edinburgh Twitter corpus. in Proceedings of the NAACL HLT 2010 Workshop on Computational Linguistics in a World of Social Media. Association for Computational Linguistics.

[19] Martín-Wanton, Tamara, Julio Gonzalo, and Enrique Amigó. "An unsupervised transfer learning approach to discover topics for online reputation management." Proceedings of the 22nd ACM international conference on Conference on information & knowledge management. ACM, 2013.

[20] Jean-Vallere Cossu, et al. "LIA@RepLab 2013"

[21] Spina, Damiano, et al. "UNED Online Reputation Monitoring Team at RepLab 2013."

[22] Berrocal, José Luis Alonso, Carlos G. Figuerola, and Ángel Zazo Rodríguez. "REINA at RepLab2013 Topic Detection Task: Community Detection."

[23] http://www.vosviewer.com/

[24] Enrique Amigó, Jorge Carrillo-de-Albornoz, Irina Chugur, Adolfo Corujo, Julio Gonzalo, Tamara Martín, Edgar Meij, Maarten de Rijke, Damiano Spina. *Overview of RepLab 2013: Evaluating Online Reputation Monitoring Systems* Proceedings of the Fourth International Conference of the CLEF initiative. 2013.

[25] Rangrej, Aniket, Sayali Kulkarni, and Ashish V. Tendulkar. "Comparative study of clustering techniques for short text documents." Proceedings of the 20th international conference companion on World wide web. ACM, 2011.

[26] Chen, Keling, and Huasha Zhao. "Twititude: Message Clustering and Opinion Mining on Twitter." (2012).

[27] Tsur, Oren, Adi Littman, and Ari Rappoport. "Scalable multi stage clustering of tagged micro-messages." Proceedings of the 21st international conference companion on World Wide Web. ACM, 2012.

[28] Tushar Khot. "Clustering Twitter Feeds using Word Co-occurrence." CS769 Project Report

[29] E. Amigo, J. Gonzalo, J. Artiles, and F. Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. Inf. Retr., 12:461–486, August 2009.

[30] migo, E., Gonzalo, J., Verdejo, F.: Reliability and sensitivity: Generic evaluation measures for document organization tasks. Technical report, UNED (2012) DATASET

[31] Overview of RepLab 2012: Evaluating Online Reputation Management Systems

[32] Fast unfolding of communities in large networks, Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre, Journal of Statistical Mechanics: Theory and Experiment 2008 (10), P1000

[33] udo Waltman, Nees Jan van Eck, and Ed C.M. Noyons (2010): A unified approach to mapping and clustering of bibliometric networks. Journal of Informetrics, 4, 629-635 vos

[34] Newman, Mark EJ. "Modularity and community structure in networks." Proceedings of the National Academy of Sciences 103.23 (2006): 8577-8582.

[35] Girvan, Michelle, and Mark EJ Newman. "Community structure in social and biological networks." Proceedings of the National Academy of Sciences 99.12 (2002): 7821-7826.

[36] Clauset, Aaron, Mark EJ Newman, and Cristopher Moore. "Finding community structure in very large networks." Physical review E 70.6 (2004): 066111.

[37] Fortunato, Santo, and Marc Barthelemy. "Resolution limit in community detection." Proceedings of the National Academy of Sciences 104.1 (2007): 36-41.

[38] Arenas, Alex, Alberto Fernandez, and Sergio Gomez. "Analysis of the structure of complex networks at different resolution levels." New Journal of Physics 10.5 (2008): 053039.

[39] Reichardt, Jörg, and Stefan Bornholdt. "Statistical mechanics of community detection." Physical Review E 74.1 (2006): 016110.

[40] Steinbach, M., Karypis, G and Kumar, V (2000). a comparison of document clustering techniques

[41] Pantel, Patrick, and Dekang Lin. "Efficiently clustering documents with committees." PRICAI 2002: Trends in Artificial Intelligence. Springer Berlin Heidelberg, 2002. 424-433.

[42] Bagga, Amit, and Breck Baldwin. "Entity-based cross-document coreferencing using the vector space model." Proceedings of the 17th international conference on Computational linguistics-Volume 1. Association for Computational Linguistics, 1998.

[43] Rosenberg, Andrew, and Julia Hirschberg. "V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure." EMNLP-CoNLL. Vol. 7. 2007.

[44] Meila, Marina. "Comparing clusterings—an information based distance." Journal of Multivariate Analysis 98.5 (2007): 873-895.

[45] Halkidi, Maria, Yannis Batistakis, and Michalis Vazirgiannis. "On clustering validation techniques." Journal of Intelligent Information Systems 17.2-3 (2001): 107-145.

[46] Levenshtein, Vladimir I. "Binary codes capable of correcting deletions, insertions and reversals." Soviet physics doklady. Vol. 10. 1966.

[47] Van Rijsbergen, C. J. (1979). *Information Retrieval* (2nd ed.). Butterworth.

[48] Vapnik, V., *Pattern recognition using generalized portrait method, in Automation and Remote Control*, 24, pp 774-780, 1963

[49] Yuan, G-X, Ho, C-H and Lin, C-J,*Recent advances of large-scale linear classification*, in proceedinngs of the IEEE, 100, 9, pp 2584-2603, 2012, IEEE

[50] Schapire, R. E., *The Boosting Approach to Machine Learning: An Overview*, in Workshop on Non-linear Estimation and Classification, 2002.

[51] Schapire, R. E. and Singer,Yoram, *BoosTexter: A boosting-based system for text Categorization, in Machine Learning*, 39, 135-168, 2000

[52] Favre, B. and Hakkani-Tür, D. and Cuendet, S., *Icsiboost: an opensource implementation of BoosTexter*, http://code.google.com/p/icsiboost, 2007

[53] Robertson,S. *Understanding inverse document frequency: on theoretical arguments for IDF*, in Journal of Documentation, 60, 5, pp 503-520, 2004, Emerald Group Publishing Limited

[54] Dong, T., Shang, W. and Zhu, H., *An Improved Algorithm of Bayesian Text Categorization*, in Journal of Software, 6, 9, pp 1837-1843, 2011

[55] Salton, G. et Buckley, C., *Term weighting approaches in automatic text retrieval*, in Information Processing and Management 24, pp 513–523, 1988.

[56] Torres-Moreno, J.-M., El-Beze, M., Bellot, P. and Bechet, F. *Opinion detection as a topic classification problem*, in Textual Information Access. Chapter 9, ISTE Ltd John Wiley and Son. 2012

[57] Anta, A.F., et al., *Sentiment Analysis and Topic Detection of Spanish Tweets: A Comparative Study of of NLP Techniques*. Procesamiento del Lenguaje Natural, 2012. 50: p. 45-52.

[58] Salton, G. and C. Buckley, *Term-weighting approaches in automatic text retrieval*. Information Processing and Management, 1988. 24(5): p. 513-523.

[59] Qureshi, M.A., C. O'Riordan, and G. Pasi. *Concept Term Expansion Approach for Monitoring Reputation of Companies on Twitter*. in CLEF (Online Working Notes/Labs/Workshop).

[60] Benhardus, J. and J. Kalita, *Streaming trend detection in Twitter*. International Journal of Web Based Communities, 2013. 9(1): p. 122-139.

[61] Osborne, M., et al. *Bieber no more: First story detection using Twitter and Wikipedia*. in SIGIR 2012 Workshop on Time-aware Information Access.

[62] Meij, E., Weerkamp, W., de Rijke, M.: *Adding semantics to microblog posts*. In: Proceedings of the fifth ACM international conference on Web search and data mining (2012)

[63] Spina, D., Meij, E., de Rijke, M., Oghina, A., Bui, M., Breuss, M.: *Identifying entity aspects in microblog posts*. In: SIGIR '12: Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval (2012)

[64] Wang, X., McCallum, A.: *Topics over time: a non-markov continuous-time model of topical trends*. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 424–433. KDD '06, ACM, New York, NY, USA (2006)

[65] Zhao, W.X., Jiang, J., Weng, J., He, J., Lim, E.P., Yan, H., Li, X.: *Comparing Twitter and traditional media using topic models*. In: Proceedings of ECIR'11. pp. 338–349. Springer-Verlag, Berlin, Heidelberg (2011)

[66] http://dev.mysql.com/doc/refman/5.5/en/fulltext-stopwords.html

[67] http://docs.oracle.com/cd/B28359_01/text.111/b28304/astopsup.htm

[68] Amigo, E., Gonzalo, J., Artiles, J., Verdejo, F.: Combining evaluation metrics via the unanimous improvement ratio and its application to clustering tasks. Journal of Artificial Intelligence Research 42(1), 689–718 (2011)

[69] https://pypi.python.org/pypi/Unidecode

[70] http://www.nltk.org/

[71] https://sites.google.com/site/findcommunities/

[72] Cilibrasi, Rudi L., and Paul MB Vitanyi. "The google similarity distance." Knowledge and Data Engineering, IEEE Transactions on 19.3 (2007): 370-383.

[73] http://www.cavar.me/damir/LID/

[74] Hassanzadeh, Oktie, Mohammad Sadoghi, and Renée J. Miller. "Accuracy of Approximate String Joins Using Grams." QDB. 2007.

[75] Kumar, Deept, et al. "Algorithms for storytelling." Knowledge and Data Engineering, IEEE Transactions on 20.6 (2008): 736-751.

[76] Chierichetti, Flavio, et al. "Finding the jaccard median." Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2010.

[77] Ioffe, Sergey. "Improved consistent sampling, weighted minhash and l1 sketching." Data Mining (ICDM), 2010 IEEE 10th International Conference on. IEEE, 2010.

[78] Wang, Jiannan, Guoliang Li, and Jianhua Fe. "Fast-join: An efficient method for fuzzy token matching based string similarity join." Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011.

[79] Zhao, Wayne Xin, et al. "Comparing twitter and traditional media using topic models." Advances in Information Retrieval. Springer Berlin Heidelberg, 2011. 338-349.

[80] Weng, Jianshu, et al. "Twitterrank: finding topic-sensitive influential twitterers." Proceedings of the third ACM international conference on Web search and data mining. ACM, 2010.

[81] Hong, Liangjie, and Brian D. Davison. "Empirical study of topic modeling in twitter." Proceedings of the First Workshop on Social Media Analytics. ACM, 2010.

[82] Ramage, Daniel, Susan T. Dumais, and Daniel J. Liebling. "Characterizing Microblogs with Topic Models." ICWSM 10 (2010): 1-1.