Master in Artificial Intelligence (UPC-URV-UB)

Master of Science Thesis

Towards Trust-aware Recommendations In Social Networks

Alberto Lumbreras Carrasco

Advisor: Ricard Gavaldà i Mestre

June 2012

Acknowledgements

Thanks to Xavier Amatriain
for inspiring the first ideas for this thesis after wondering, upset,
why *last.fm* hadn't recommend him that upcoming concert
of his favorite and most listened band, Los Planetas;
for pointing me to *Trust-aware Recommender Systems* when I was lost,
and for introducing me the *Recommender Systems Handbook*.

Thanks to the authors of the *handbook*
for writing and compiling such a comprehensive
introduction to the state of the art of recommender systems.
This book has been specially helpful when writing
Chapter 2 of this Master Thesis and has served
as a starting point to find further references.

Thanks to Elisa for her laughs,
the strength and the motivation,
and for having taken too much care of our *reproductive labour*.

Thanks to my parents for their dedication.
This Thesis wouldn't have been possible without
of all the efforts they invested on me.

Thanks to my Master Thesis Advisor, Ricard Gavaldà,
for opening the doors of research to me,
and allowing me to pursue that old human dream
of combining work and pleasure.

Thanks to all the men and women
that struggled to get a public education system,
and to those who are still struggling to defend it today.

A David

Abstract

Recommender systems have been strongly researched within the last decade. With the emergence and popularization of social networks a new field has been opened for social recommendations. Introducing new concepts such as trust and considering the network topology are some of the new strategies that recommender systems have to take into account in order to adapt their techniques to these new scenarios.

In this thesis a simple model for recommendations on twitter is developed to apply some of the known techniques and explore how well the state of the art does in a real scenario. The thesis can serve as a basis for further social recommender system research.

Contents

Chapter 1

Introduction

1.1 The paradox of choice

The explosive growth and variety of information available on the Web and the rapid introduction of new services (blog feeds, buying products, product comparison, auction, etc.) frequently overwhelmed users, leading them to make poor decisions. The availability of choices, instead of producing a benefit, started to decrease users' comfort. It was understood that while choice is good, more choice is not always better. Indeed, choice, with its implications of freedom, autonomy, and self-determination can become excessive, creating a sense that freedom may come to be regarded as a kind of misery-inducing tyranny [26]

With the recent emerging of social networks internet users became directly interconnected to share all kind of information (statuses, professional profiles, videos, news, etc). Users connect to other users to share information between them. As the number of friends grows, the amount of information received by a user will grow proportionally. This aggravate the information overload problem as users are passive receivers of information generated by their set (sometimes hundreds) of friends.

Social networks are like rooms where everybody is talking. Someone, or somewhat, has to advice users on who to at every moment. Or even bring the user some piece of interesting information that has been said in other room. Information with no interest for the user should be muted, and high value information should be showed in first place.

1.2 Recommending in social networks

Recommender Systems are software tools and techniques providing suggestions for items of interest to a user. The suggestions provided are aimed at supporting their users in various decision-making processes, such as what items to buy, what music to listen, or what news to read. Recommender systems have proven to be valuable means for online users to cope with the information overload and have become one of the most powerful and popular tools in electronic commerce. Correspondingly, various techniques for recommendation generation have been proposed and during the last decade, many of them have also been successfully deployed in commercial environments.

Evidence suggests that people tend to rely more on recommendations from their friends than on recommendations from similar but anonymous individuals [27]. Thus, it makes sense to have the ratings of a recommender system influenced by the ratings of the user's friends.

Recommending systems on social networks (known as Social Recommender Systems) face three main challenges: first, they have to learn to rank the information coming from a user's friends. Second, they have to discover information from sources other than a user direct friends. Third, they have to decode information provided by the structure of the network (friend relationships) and user interactions (i.e.: trust).

Most recommendation techniques consider that the system has all the potentially useful information (items, users and ratings). Amazon stores in its databases all the information about available books, registered users, and past interactions such as purchases or book ratings. Facebook can place personalized ads into a user interface based on complete knowledge of the user's activity in Facebook. But sometimes we have only a limited access to the data, since we are not the owners. In these cases, we must use some crawling technique to get those subsets of data that will be more useful to the task at hand.

1.3 Aims of the thesis

Social recommender systems need new techniques to deal with recommendations on social network scenarios. Several contributions have been made to the field. One of the aims of this thesis is to contribute to the discussion of how to compute trust from interactions on a social network where there is no explicitly annotated information. This thesis explores how to tackle this issue in the Twitter[1] social network.

The desired utility of our trust metric is to enhance recommendations of tweets. To evaluate trust-based recommendations we propose an architecture for social recommendations. Here we explore different techniques at three different levels: crawling, trust propagation and text mining. Crawling aims to get the optimum set of neighbors for every user, in such a way that the value of items published in this neighborhood is maximized. Trust and trust propagation aim to compute trust between pairs of users and use this information to enhance recommendations with social information rather than just using content-based recommendations. Text mining focuses on how to encode tweets and how to compute similarities to use in a content-based recommendation of tweets. We explore the state of the art techniques that can be useful in a such a scenario and study how to put them together to build a recommender system that filters and finds useful information for the user.

The main goal is not to get astonishing results astonishingly precise results for recommendation, as this is likely to be difficult with partial information, but to get a first intuition on what pieces of existing technologies and what techniques are more suitable for recommending items in social networks. The thesis can serve as a basis for further social recommender system research.

A paper on the results of this thesis have been accepted for presentation at the ASONAM 2012[2] workshop "International Workshop of Social Knowledge Discovery and Utilization"[3]

[1] www.twitter.com

[2] http://www.asonam2012.etu.edu.tr/

[3] http://people.cs.aau.dk/ rpan/skdu/

Chapter 2

Background

Recommender Systems typically apply techniques from other neighbor areas. The most important of these areas are data mining and machine learning, which provide the core methods of a recommender. This chapter is an overview of the main techniques that are to be considered when designing a recommender system. The sections that follow do not correspond to the real classification of areas (Text Mining can be considered a subarea of Data Mining or Natural Language Processing , and classification techniques are in between Machine Learning and Data Mining. Even the relationship between Data Mining and Machine Learning is a matter of discussion into which we do not want to enter in this document). However, they are grouped as toolboxes, where a toolbox would contain the techniques necessary to face a part of the recommendation problem (preprocessing, machine learning, dealing with textual items, and global strategy of the recommender).

2.1 Data mining

In this section two different problems are explained: the first one is how to compare items. No matter whether the recommender strategy is to look at what *similar* users did in the past (collaborative filtering) or what are *similar* items to the ones that a user bought (content based) at some point we will need to choose a similarity measure or make our own. The second one is reduction of dimensionality. The curse of dimensionality is a well known problem in machine learning. Every item is represented as a set of features that can be considered as a point in a n-dimensional space, where n corresponds to the number of features that we use to represent the items. As the number of features (dimensions) increases, the vector space gets more sparse and it makes some tasks (i.e.: clustering) harder. Note that when dealing with text items, where often every word corresponds to a different feature, this problem gets worse.

2.1.1 Similarity measures

Recommender systems usually face a problem of computing similarity between users (or items). The more similar two users are, the more likely it is that a new item liked by one of these users is going to be liked by the other. Different similarity measures (distances) can be applied to a problem, and the best choice depends on the scenario. One of the most common similarity measure is Euclidean Distance:

$$d(x,y) = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2} \tag{2.1}$$

Euclidean distance can be seen as an specific form of the generalized Minkowsky distance with $r = 2$ (L_2-norm). Minkowsky distance is a generalized metric for Euclidean spaces with this generalized form:

$$d(x,y) = (\sum_{k=1}^{n} |x_k - y_k|^r)^{\frac{1}{r}} \tag{2.2}$$

With $r = 1$ we get a Manhattan distance, or L_1.

$$d(x,y) = \sum_{k=1}^{n} |x_k - y_k| \tag{2.3}$$

With $r = \infty$ we get a Tchebychev distance, Maximum Metric, or L_∞, where the distance between two vectors is the greatest of their differences along any dimension:

$$d(x,y) = \max_k |x_k - y_k| \tag{2.4}$$

Another common approach is consider distance as the cosine of the angle between two vectors of attributes.

$$cos(x,y) = \frac{xy}{||x||||y||} \tag{2.5}$$

Cosine distance is useful when items are documents represented as bags-of-words. As it only considers the angle between the documents, term vectors can be normalized to the unit sphere for more efficient processing.

Pearson correlation is useful when we need to avoid rating inflations. Imagine we have a movie recommender systems. We can expect two users with same preferences rate a set of movies in a similar way. But if one user is more "optimistic" her ratings will be a bit inflated with respect to the other. To compensate this inflation and focus only on the normalized similarity between two user ratings, we can use Pearson correlation:

$$Pearson(x,y) = \frac{\sigma_{xy}}{\sigma_x \sigma_y} \tag{2.6}$$

Notice that Pearson correlation does not work if one of the users gives to every item the same rating, as that implies $\sigma = 0$.

Finally, Mahalanobis distance is similar to Euclidian distance, but it introduces the covariance matrix σ into the formula:

$$d(x,y) = \sqrt{(x-y)\sigma^{-1}(x-y)^T} \tag{2.7}$$

Deciding which similarity distance to use depends on the scenario and we will usually have to test different metrics.

2.1.2 Reducing dimensionality

It is common in data mining to have instances with a large set of attributes each, which means that instances live in a sparse high-dimensional space. Instances are very far from each other, and that makes it hard for machine learning algorithms to learn similarity patterns as instances can be close in some dimensions but far in others. This is called the *curse of dimensionality*. To overcome this problem different dimensionality reduction techniques can be applied to the dataset. Some of them will directly find non-relevant attributes to delete (feature selection). Others will perform this by first projecting the space into a less sparse one and then discarding those dimensions where less information is contained.

In the following sections, we will review two of theses projection techniques, known as Principal Component Analysis (PCA) and Single Value Decompositions (SVD).

Principal Components Analysis

Principal Component Analysis (PCA) is an statistical method based on the following assumption: the more variance a dimension contains, the more informative it is. PCA projects the original space into a new one that maximizes the variance of the first dimensions (or components). Thus, the first component is the most informative, the second component is the second most informative, and so on. We can reduce the dimensionality of the data by neglecting the less informative components. A rule of thumb threshold is used to decide which dimension we should start neglecting from. Note that PCA creates a linear combination of the original space, that is, it performs a rotation of the axis.

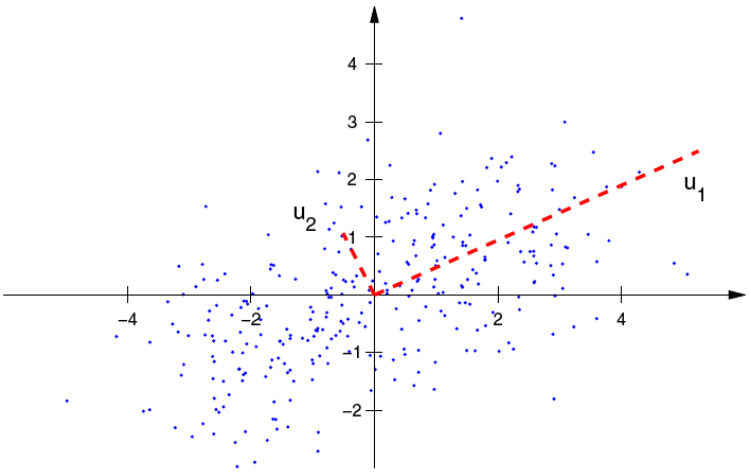

Figure 2.1: PCA analysis of a two-dimensional set of points. The principal components derived are u_1 and u_2. The variance contained in each component is relative to the information that it contains. (source: [22][p. 45])

Single Value Decomposition

Single Value Decomposition (SVD) [10] is a matrix factorization method that decomposes a matrix M to a form:

$$M_{n\times m} = U_{n\times r}\Sigma_{r\times r}V^T_{r\times n} \tag{2.8}$$

where U and V are orthonormal basis and Σ is a non-negative diagonal matrix. The diagonal values of Σ are known as the *singular values* of M and are sorted in decreasing order. By applying Single Value Decompositions we find a dimensional feature space where the new features represent "concepts" and the strength of each concept in the context of the collection is computable. After performing a Single Value Decomposition to a matrix M we can reduce the dimensionality of M by truncating the singular values (as well as U and V^T) at a given k. The truncated SVD is a representation of the underlying latent structure in a reduced k-dimensional space, which generally means that the noise in the features is reduced.

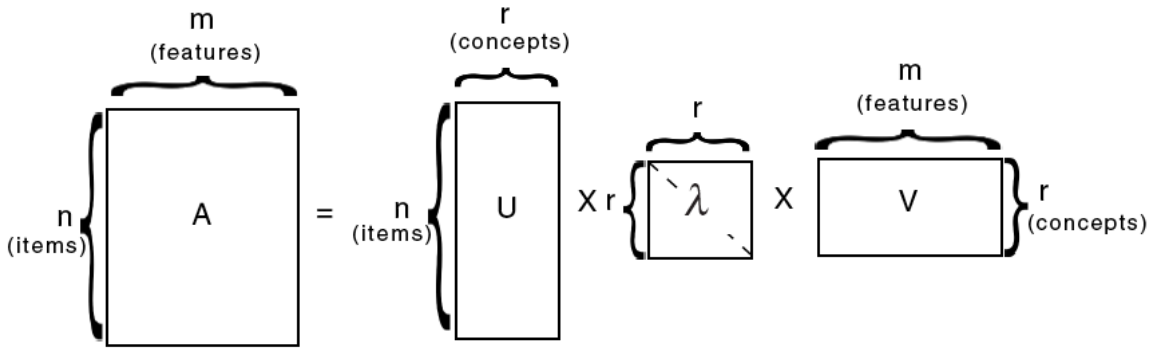

Figure 2.2: An illustration of a Singular Value Decomposition: an item × features matrix can be decomposed into three different ones: an item × concepts, a concept strength, and a concept × features (source: [22] p. 46)

The main difference with PCA, and its advantage, is that SVD not only performs a rotation but an scaling as well. Because SVD allows to automatically derive semantic "concepts" in a low dimensional space, it can be used as the basis of latent-semantic analysis, a popular technique for text classification.

2.2 Classification

In this section we review some of the most important classification techniques often applied to text.

Nearest Neighbors

Nearest neighbors classifier (kNN) [5] finds the k closest points to the one to be classified, and then assigns the class label considering the class labels of these nearest-neighbors. The underlying idea is that if a record falls in a particular neighborhood where a class label is predominant it is because the record is likely to belong to that same class. More formally, and implicitly, kNN is a non-parametric density estimator, that is, a method that infers the probability density function behind each one of the classes that generated the population.

Knowing these densities and given a new instance it is straightforward (using Bayes' theorem and considering the common approximation for density estimation $p(x) \approx \frac{(K/n)}{V}$ where V is the volume surrounding x, n is the total number of examples and K is the number of neighbors inside V) a the volume that contains K points is the number of points inside) to calculate the probability of this instance belonging to each class. Given K_i neighbors belonging to class C_i from a total number of K neighbors, the probability of class membership can be expressed as:

$$p(C_i|x) = \frac{p(x|C_i)p(C_i)}{p(x)} = \frac{p(x, C_i)}{\sum_{j=1}^{m} p(x, C_j)} \approx \frac{K_i/n}{V \sum_{j=1}^{m} \frac{K_j/n}{V}} = \frac{K_i}{\sum_{j=1}^{m} K_j} = \frac{K_i}{K} \tag{2.9}$$

thus we can classify a new instance by a majority vote of its neighbors, with the instance being assigned to the predominant class amongst its K nearest neighbors.

Nearest neighbors can be also used for regression, by simply assigning the property value for the object to be the average of the values of its k nearest neighbors. Nearest Neighbors is the base algorithm underlying Collaborative Filtering recommendations.

Decision Trees

Decision trees are classifiers in the form of tree structures. In its simplest form, a decision tree is a tree in which each non-leaf node denotes a test on an attribute of cases, each branch corresponds to an outcome of the test, and each leaf node denotes a class prediction (see Figure 2.3). Although every non-leaf node is created after a decision on the value of a single attribute, these nodes can also bee seen the conjunction of the features (and their values) of the parent nodes. Because of this, decision trees are a very transparent and visual model to understand how instances are classified. Most decision tree induction algorithms work by splitting the original set of instances into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner. This process stops once all observations belong to the same class to the same class, or some other impurity criterion is satisfied. For practical reasons, however, most decision trees implementations use pruning by which a node is no further split if its impurity measure or the number of observations in the node are below a certain threshold. There are many algorithms for decision tree induction: CART, ID3 and C4.5 to mention the most common. Decision trees may be used in a model or in a content based approach for a recommender system.

Bayesian Classifiers

A Bayesian classifier [7] is a probabilistic classifier based on parametric density estimations. It is based on the definition of conditional probability and the Bayes theorem. The probability of a class given the data (posterior) is proportional to the product of the likelihood times the prior probability of the class (or prior). The prior specifies the *a priori* belief in the model before the data was observed. The goal is to find the class C_k that maximizes the posterior probability.

The probability of an item d being in class c is computed as:

$$P(c|d) \propto P(d|c)P(c) = P(t_1, ..., t_k|c)P(c) \tag{2.10}$$

where t_k is the k-est attribute of instance d. To simplify the computation of the model, a very common assumption is made of assuming the probabilistic independence of the attributes.

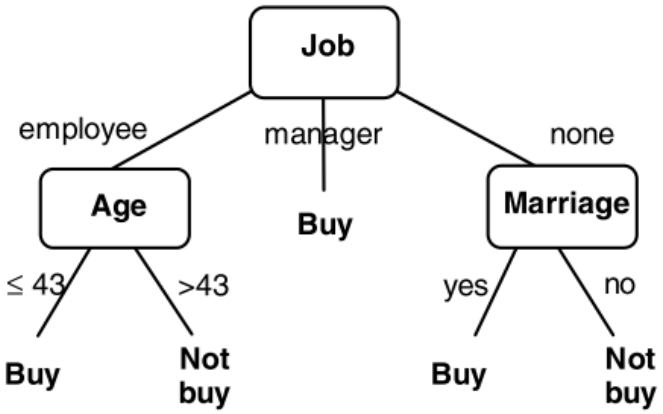

Figure 2.3: An example of a decision tree for purchase assessment

When this assumption is made the model is called a *Naive Bayes*, and the formulation is:

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c) \tag{2.11}$$

Bayesian classifiers are particularly popular for content-based recommender systems.

Support Vector Machines

The goal of a Support Vector Machine (SVM) classifier [4] is to find a linear hyperplane (decision boundary) that separates positive and negative instances in such a way that the margin is maximized. For instance, if we look at a two class separation problem in two dimensions like the one illustrated in figure, we can easily observe that there are many possible boundary lines to separate the two classes. Each boundary has an associated margin. The rationale behind SVM's is that if we choose the one that maximizes the margin we are less likely to misclassify unknown items in the future. Given a training set, it seems that we would found a good fit to the training data if we can find a vector w so that $w^T x^{(i)} \gg 0$ whenever $y^{(i)} = 1$ and $w^T x^{(i)} \ll 0$ whenever $y^{(i)} = 0$ since this would reflect a very confident set of classifications for all the training examples.

Let $(x^{(i)}, y^{(i)})$ be an input feature vector and its label. SVM proposes to find a vector w such that:

$$y^{(i)}(w^T x^{(i)} + b) \geq 0 \tag{2.12}$$

that is, we want to find a vector such that makes the product $(w^T x^{(i)} + b)$ to be of the same sign that $y^(i)$. Moreover, the greater this product is, the more confident we will be. This product is the *functional margin*. The functional margin of the training set is the minimum functional margin of its samples. Note that if we increase the size of the hyperplane to $2w$ and $2b$ the final classification will remain unchanged while the functional margin will artificially increase. As we are can arbitrarily increase the confidence of the classification it does not seem a good idea. To avoid this, we normalize w. Now we can pose the following optimization

problem:

$$\max_{w,b} \gamma \quad \text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \quad \forall i$$
$$||w|| = 1$$

Solving this equation problem is not trivial and requires to transform it to a Quadratic

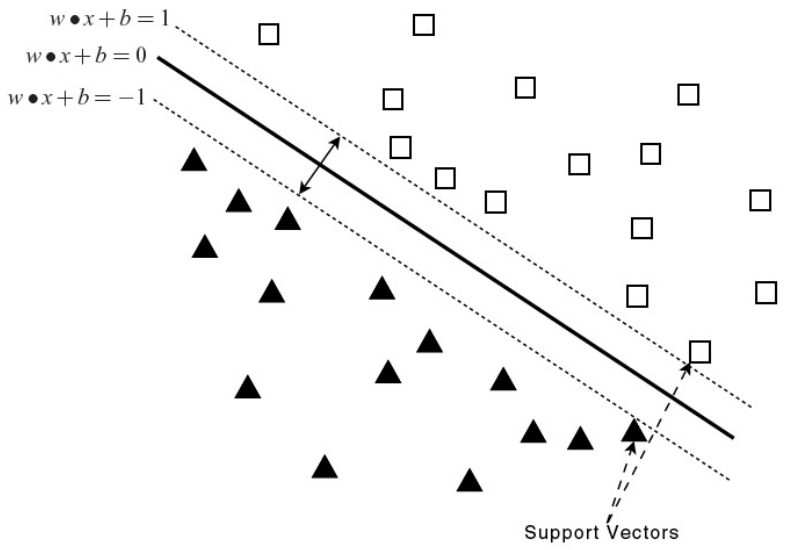

Figure 2.4: Illustration of support vectors (source: [22][p. 56])

Programming problem.

SVM are very popular to classify text, as they are very robust under high dimensional documents. However, if the input set is not linearly separable the SVM classifier will fail. Kernel methods try to solve this problem. The idea of Kernel methods is to map the input features to a new space where the instances are linearly separable. If this new space satisfies some conditions there will be a duality between both spaces and thus classifications in the second space will still be valid in the original one. We omit details here.

2.3 Text mining

Vector space models

The Vector Space Model is the basic methodology proposed by Information Retrieval researchers for representing text corpora. It reduces each document in the corpus to a n-dimensional space $d_j = \{w_{1j}, w_{2j}, ..., w_{nj}\}$ where each dimension w_i corresponds to the degree of association between the document and the corresponding term from an overall vocabulary $T = \{t_1, t_2, ..., t_n\}$. The vocabulary is built from the document collection by applying some standard operations, such as tokenization, stopwords removal, and stemming. To limit the number of dimensions the vocabulary is often truncated by dismissing the less frequent words.

The value of each dimension is a real number that can be computed in several ways. In the popular *tf-idf* scheme [25], for each document in the corpus, a count is formed of the number of occurrences of each word (term frequency). After suitable normalization, this term frequency count is multiplied by an inverse document frequency count, which measures the number of occurrences of a word in the entire corpus (generally on a log scale, and again suitably normalized). The end result is a term-by-document matrix whose columns contain the *tf-idf* values for each of the documents in the corpus. *tf-idf* identifies how discriminative is a word in a document, which makes it useful for search engines.

Latent Semantic Analysis

While vector space techniques have some appealing features - e.g.: when used with *tf-idf* to identify discriminative words- they provide a small capability for reducing dimensionality and reveal little about inter or intra document statistical structure. To address these shortcomings, some techniques such as Latent Semantic Analysis (or Indexing) have been proposed.

The key idea of Latent Semantic Analysis (LSA) [16] is to map high-dimensional count vectors, such as the ones arising in vector space representations of text documents, to a lower dimensional representation in a so-called latent semantic space. As the name suggests, the goal of LSA is to find a data mapping which provides information well beyond the lexical level and reveals semantical relations between the entities of interest. Due to its generality, LSA has proven to be a valuable analysis tool with a wide range of applications

The information derived by LSA are not simple frequencies or co-occurrence counts, but "latent semantic" concepts that are often much better predictors of human meaning than are other surface level contingencies. A way to think of LSA is that it represents the meaning of a word as a kind of average of the meaning of all the passages in which it appears, and the meaning of a passage as a kind of average of the meaning of all the words it contain.

To compute LSA in text, the first step is to represent the text as a matrix in which each row stands for a unique word and each column stands for document. Each cell contains the word frequency in that document (see original text in Figure 2.5 and its matrix representation in Figure 2.6). Sometimes it is good to apply a preprocess to these cells to reflex a more relevant information rather than the simple word count . Next, LSA applies singular value decomposition (SVD) to the matrix. In SVD, a rectangular matrix is decomposed into the product of three other matrices (see Figure 2.7). One can reduce the dimensionality of the solution simply by deleting smallest coefficients in the diagonal matrix. After reducing dimensionality we can reconstruct the original space by multiplying the truncated factors (see Figure 2.8 that shows the reconstruction of the original matrix after truncating its factorized form after the second component). What we have is a matrix that looks like the original one but with smoothed values. We expect this new matrix to capture more semantic information than the original one.

Latent Dirichlet allocation

LDA [2] is a bayesian generative model based on three hierarchical levels. Each item of a collection is modeled as a finite mixture over an underlying set of topics. Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities. LDA was first presented as a graphical model for topic discovery. It allows sets of observations to be explained by unobserved variables that explain the word composition of the document. For

Example of text data: Titles of Some Technical Memos

c1: *Human* machine *interface* for ABC *computer* applications
c2: A *survey* of *user* opinion of *computer system response time*
c3: The *EPS user interface* management *system*
c4: *System* and *human system* engineering testing of *EPS*
c5: Relation of *user* perceived *response time* to error measurement

m1: The generation of random, binary, ordered *trees*
m2: The intersection *graph* of paths in *trees*
m3: *Graph minors* IV: Widths of *trees* and well-quasi-ordering
m4: *Graph minors*: A *survey*

Figure 2.5: Example of documents set

$\{X\} =$

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

Figure 2.6: Term frequency matrix

example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's creation is attributable to one of the document's topics.

Each topic is a distribution over words. Each document is a mixture of topics. Each word is drawn from one of these topics. Figure 2.9 shows how this layers affect the content of the final document. But as shown in Figure 2.10, in reality, we only observe the document and its words. The other structures are hidden variable that must be inferred.

In LDA we must explicitly choose two parameters. The Dirichlet parameter controls the shape of the distributions and hence the likelihood of a topic being selected. Each topic has also its own parameter, a greater value implying a more probable topic. The exchangeable Dirichlet distribution requires all parameters to be equal, leading to a set of topics having the same likelihood for all topics. Small values lead to only a few topics allocated to each document. In the other hand, we need to infer the per-word topic assignment $z_{d,n}$, the per-document topic proportions θ_d and the per-corpus topic distributions β_k.

Figure 2.11 shows the graphical model of LDA.

$$\{X\} = \{W\}\{S\}\{P\}'$$

$$\{W\} =$$

0.22	-0.11	0.29	-0.41	-0.11	-0.34	0.52	-0.06	-0.41
0.20	-0.07	0.14	-0.55	0.28	0.50	-0.07	-0.01	-0.11
0.24	0.04	-0.16	-0.59	-0.11	-0.25	-0.30	0.06	0.49
0.40	0.06	-0.34	0.10	0.33	0.38	0.00	0.00	0.01
0.64	-0.17	0.36	0.33	-0.16	-0.21	-0.17	0.03	0.27
0.27	0.11	-0.43	0.07	0.08	-0.17	0.28	-0.02	-0.05
0.27	0.11	-0.43	0.07	0.08	-0.17	0.28	-0.02	-0.05
0.30	-0.14	0.33	0.19	0.11	0.27	0.03	-0.02	-0.17
0.21	0.27	-0.18	-0.03	-0.54	0.08	-0.47	-0.04	-0.58
0.01	0.49	0.23	0.03	0.59	-0.39	-0.29	0.25	-0.23
0.04	0.62	0.22	0.00	-0.07	0.11	0.16	-0.68	0.23
0.03	0.45	0.14	-0.01	-0.30	0.28	0.34	0.68	0.18

$$\{S\} =$$

3.34								
	2.54							
		2.35						
			1.64					
				1.50				
					1.31			
						0.85		
							0.56	
								0.36

$$\{P\} =$$

0.20	0.61	0.46	0.54	0.28	0.00	0.01	0.02	0.08
-0.06	0.17	-0.13	-0.23	0.11	0.19	0.44	0.62	0.53
0.11	-0.50	0.21	0.57	-0.51	0.10	0.19	0.25	0.08
-0.95	-0.03	0.04	0.27	0.15	0.02	0.02	0.01	-0.03
0.05	-0.21	0.38	-0.21	0.33	0.39	0.35	0.15	-0.60
-0.08	-0.26	0.72	-0.37	0.03	-0.30	-0.21	0.00	0.36
0.18	-0.43	-0.24	0.26	0.67	-0.34	-0.15	0.25	0.04
-0.01	0.05	0.01	-0.02	-0.06	0.45	-0.76	0.45	-0.07
-0.06	0.24	0.02	-0.08	-0.26	-0.62	0.02	0.52	-0.45

Figure 2.7: SVD Factorization of the original matrix

$$\left\{\hat{X}\right\} =$$

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	0.16	0.40	0.38	0.47	0.18	-0.05	-0.12	-0.16	-0.09
interface	0.14	0.37	0.33	0.40	0.16	-0.03	-0.07	-0.10	-0.04
computer	0.15	0.51	0.36	0.41	0.24	0.02	0.06	0.09	0.12
user	0.26	0.84	0.61	0.70	0.39	0.03	0.08	0.12	0.19
system	0.45	1.23	1.05	1.27	0.56	-0.07	-0.15	-0.21	-0.05
response	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
time	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
EPS	0.22	0.55	0.51	0.63	0.24	-0.07	-0.14	-0.20	-0.11
survey	0.10	0.53	0.23	0.21	0.27	0.14	0.31	0.44	0.42
trees	-0.06	0.23	-0.14	-0.27	0.14	0.24	0.55	0.77	0.66
graph	-0.06	0.34	-0.15	-0.30	0.20	0.31	0.69	0.98	0.85
minors	-0.04	0.25	-0.10	-0.21	0.15	0.22	0.50	0.71	0.62

Figure 2.8: Final recomposition (source [15])

2.4 Recommender systems

With the popularization of e-commerce and many internet services where users "consume" items such as news, blogs or music, appeared the demand of new services that help users to

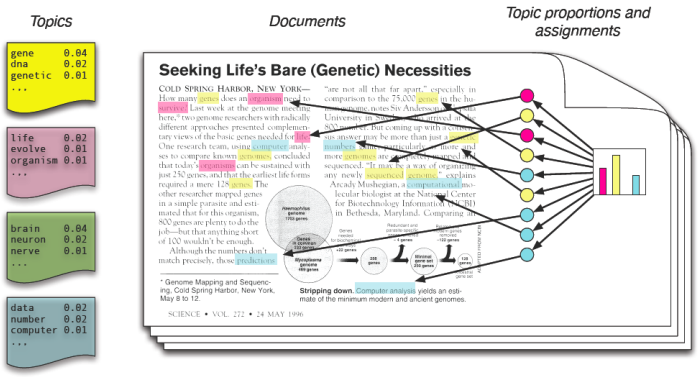

Figure 2.9: Illustration of support vectors

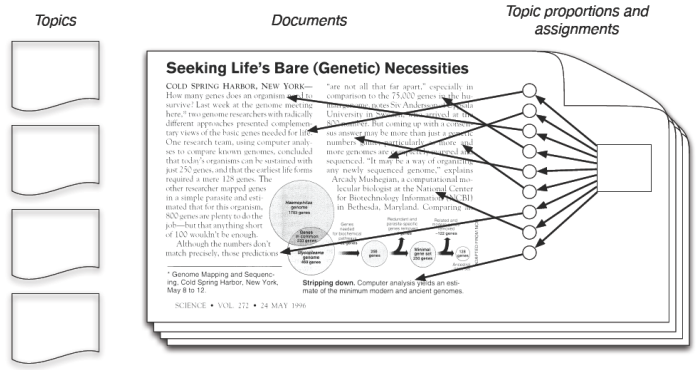

Figure 2.10: Illustration of support vectors

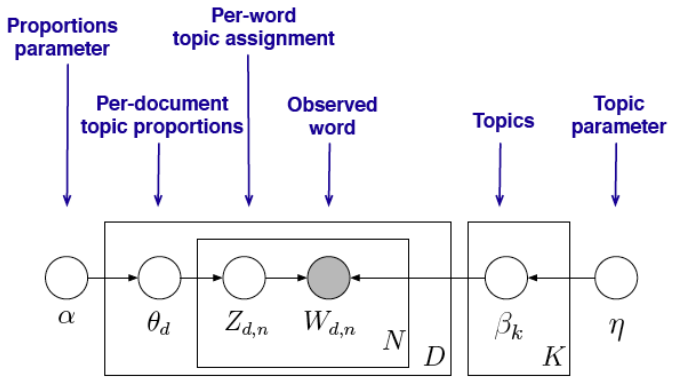

Figure 2.11: Illustration of support vectors

decide between a overflow of choices. From the point of view of the provider, recommendation engines can increase user satisfaction and company benefits by increasing the number of purchases.

In order to implement its core function, identifying the useful items for the user, a recommender system must predict whether an item is worth recommending. To do this, the system must be able to predict the utility of the items, or at least compare the utility of some items,

and then decide what items to recommend based on this comparison.

Recommender systems emerged a decade ago and many techniques have been proposed and tested in different real scenarios to recommend different kinds of items that range from people to songs. In the next sections a brief description is given on the main families of recommendation techniques: collaborative filtering, content-based and trust-based. In collaborative filtering the algorithm bases its recommendations on other users preferences. These methods are easy to implement and require no knowledge on the items. Content-based techniques look at some features of the items to understand deeper what is what makes a user like or dislike an item. It requires a previous feature extraction that sometimes has to be done by experts (e.g Pandora recommend songs based on user feedback and song features extracted by a group of musicians). Finally, trust-based recommendations are applied when a social network or some sort of interactions between users can be inferred. These systems tune classic techniques adding trust information which is explicitly given by users or implicitly computed by trust computation algorithms.

2.4.1 Collaborative Filtering recommenders

Collaborative filtering (CF) is a popular method that makes recommendations based on similarity between users. Similarity is here measured in terms of the distance between two user vectors. A user vector contains opinions on every item. It can be boolean (liked it or not, bough it or not) or numeric (item ratings). Various distance metrics can be applied, e.g. Pearson correlation or cosine similarity. Collaborative filtering provides the user with the "best bets" she will most likely be interested in, either one single item or a "ranked list of items" – usually referred as top-N items.

Recommender systems based on collaborative filtering can provide the user with unexpected but fitting recommendations that do not have anything in common with the other rated items. Unlike content-based approaches, which use the content of items previously rated by a user u, collaborative (or social) filtering approaches rely on the ratings of u as well as those of other users in the system. The key idea is that the rating of u for a new item i is likely to be similar to that of another user v, if u and v have rated other items in a similar way. Likewise, u is likely to rate two items i and j in a similar fashion, if other users have given similar ratings to these two items.

Collaborative approaches overcome some of the limitations of content-based ones. For instance, items for which the content is not available or difficult to obtain can still be recommended to users through the feedback of other users. Finally, unlike content-based systems, collaborative filtering ones can recommend items with very different content, as long as other users have already shown interest for these different items. This facilitates the exploration of items in what is known as "the Long Tail" [1].

Collaborative filtering methods are commonly grouped in two general classes: memory and model-based methods. In memory-based collaborative filtering (also known as "lazy learning") the user-item ratings stored in the system are directly used to predict ratings for new items without previous training. The model-based approach, on the other hand, first builds a model out of the user-item interaction database and then uses this model to make recommendations.

Memory based

Memory based models are mostly based on neighborhood ratings. These can be either user-based or item-based. User-based methods predict the rating r_{ui} of a user u for a new item i using the ratings given to i by users most similar to u, called nearest-neighbors. Suppose we have for each user $v \neq u$ a value w_{uv} representing the preference similarity between u and v. The k-nearest-neighbors (k-NN) of u are the k users with the highest similarity to u that have rated item i. We denote these users by $\mathcal{N}_i(u)$. The predicted rating r_{ui} can be estimated as the average rating given to i by these neighbors. More similar neighbors should be weighted more. We can write this prediction as:

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv} r_{vi}}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|} \tag{2.13}$$

where similarity w_{vu} can be computed by methods like the ones described in Section 2.1.1.

Users rate items from a personal point of view, and there is no a absolute scale. E.g, rating movies on a 0-5 scale, a user u can consider a 3 to be an average movie and a user b can consider a 3 to be a good movie but not a masterpiece. To normalize this differences one can use Pearson correlation as similarity measure. Another popular approach is considering the mean-centered prediction:

$$\hat{r}_{ui} = \overline{r}_u + \frac{\sum\limits_{v \in \mathcal{N}_{i(u)}} w_{uv}(r_{vi} - \overline{r}_v)}{\sum\limits_{v \in \mathcal{N}_{i(u)}} |w_{uv}|} \tag{2.14}$$

The same methods can be applied swapping users and item. Note that, apart from changing the perspective and maybe the final outcome of the recommender, our data will become more or less sparse and will increase or decrease the time of computations.

Applying collaborative filtering to classification instead of regression is straightforward. Now the task is to find the class that maximizes:

$$v_{ir} = \sum_{v \in (N)_{i(u)}} \mathcal{I}(r_{vi} = r) w_{uv} \tag{2.15}$$

where $\mathcal{I}$ is the identity function and is 1 when the expression is true and 0 otherwise.

Model based

In contrast to neighborhood-based systems, model-based approaches use the user-item matrix to learn a predictive model for every user. The general idea is to use columns of the matrix as attributes and the one corresponding to the target user as output. The model is trained using those items that have been rated by the target user, and later used to predict ratings of for new items. Model-based approaches can use some classification (or regression) methods such as Bayesian Clustering [3] or Support Vector Machines [11].

2.4.2 Content-based recommenders

Content-based recommendation systems learn from some features of the items to recommend similar items to those liked from the target user in the past. Features can be automatically or

manually extracted. When manually extracted, these systems need a group of experts in the field (e.g: music) to decide which features are the most relevant ones and to manually fulfill them for every item. Items are then represented as feature vectors, from where the system can create a profile for the target user. The use of machine learning to model the user profile is very popular. This way, the tasks can be mapped into a classical machine learning task with a set of items and their rating.

When items are textual, capturing the information laying in the text is a big challenge and simple bag-of-words is often not enough, as the model is superficial and does not capture most of the semantics on the text. Sections 2.3 shows some of the main techniques to deal with text, although other approaches such as the use of ontologies are also popular.

A technique to compute user profiles when items are textual, used in several content-based recommender systems, is the Rocchio algorithm [23], a technique to get relevance feedback. This technique refines the user profile incrementally every time this user rates a new text item. The algorithm creates a prototype vector $\vec{c_i} =< w_{1i}, ...w_{Ni} >$ for every class where words are weighted according to its positive or negative relevance in the examples:

$$w_{ki} = \beta \sum_{d_j \in POS_i} \frac{w_{kj}}{|POS_i|} - \gamma \sum_{d_j \in NEG_i} \frac{w_{kj}}{|NEG_i|} \tag{2.16}$$

where ω_{kj} is the TF-IDF weight of the term k in document d_j, POS_i and NEG_i are the set of positive and negative examples in the training set for the specific class c_j, β and γ are control parameters that set the relative importance of positive and negative examples.

Content-based filtering provides some advantages compared to collaborative filtering. First, when there is no much information on other users preferences (neighbors) the quality of the recommender is not affected. Second, depending on how we process the content and the recommendations (e.g: decision trees) the system can provide an explanation on why an item is being recommender. Third, a new item can be recommended immediately even if any user has rated it, as soon as its features are extracted. On the other hand, content-based recommender systems get easily stuck in "neighbor" items and do not find items that, even if different, the target user may like. Finally, feature extraction, while powerful, is a critical task that will have an important impact on the quality of the recommendations, and domain knowledge is often needed.

2.4.3 Trust-based recommenders

Trust-based recommender systems (also known as trust-aware, social or community-based recommender systems) follow the epigram "Tell me who your friends are, and I will tell you who you are". Evidence suggests that people tend to rely more on recommendations from their friends than on recommendations from similar but anonymous individuals [27]. This observation, combined with the growing popularity of open social networks, is generating a rising interest in community-based systems or, as or as they usually referred to, social recommender systems. This systems acquire information about the social relations of the users and create a trust model that shapes the observed information. The recommendation takes into account the trust between users and can be combined with collaborative filtering or content-based filtering strategies. Trust can be used, in a similar way to that of other similarity measures, to weight the contributions of neighbor users and the more a user a trusts on user b the more ratings of b will be considered to recommend items to a.

Trust can be explicitly or implicitly acquired. When explicitly acquired, users are asked to build a Web of Trust (WOT), that is, a list of users they trust. They can even rate every user in their WOT. This is the solution taken by the movie recommendation system FilmTrust [9]. Users of FilmTrust are asked to rate friends in a 0-10 scale. For non-rated of not directly connected users a trust propagation algorithm is used to estimate trust between them and the target user.

Sometimes we do not want to bother users asking for their WOT, or we do not have this possibility. In these cases we have to find a way to estimate trust. As in social networks we have some sort of information on friendship, or interaction between users, we can try to define trust based on this relationships or interactions. We can define trust, for instance, as a normalized factor proportional to the number of interactions between two users. Trust computation changes from one social network to another. In LinkedIn, a measure of trust can be defined from endorsements and connections. In Twitter, trust can be inferred from mentions between users, and/or from the number of times that one user *retweets* another user.

To compute trust between not connected users a model of trust propagation is needed. Trust propagation models tend to apply a transitivity hypothesis: if user a trusts on user b and user b trusts on user c it can be assumed that user a trusts on user c. Another issue is how exactly this trust passes from a to c is to be defined. We can decide trust of user a on user c t_{ac} as the minimum trust on the path that connects a to b (an edge on the path can be created by a friendship or an interaction). Or maybe we prefer defining trust t_{ac} as the product of trusts on the path. When multiple paths connect a to b we can use aggregator operators. For instance, for two paths that connect a to b we can compute the total trust as the average of trusts for these two paths, the minimum, or the maximum.

Trust can be either general (for any topic) or specific (for a set of topics). In cases where trust is explicitly asked to users, dealing with topic-based trusts is straightforward, as we can consider an implicit and separate social network for every topic. However, when trust have to inferred for every topic separately then new difficulties arise. If a trusts on b on topic t and b trusts c on topic q it is probably wrong to infer that a trusts c on any topic. In this cases it is necessary to apply topic detection methods to extract active topics from the interactions (or from the item). Notice that even when talking about the same topic transitivity might not hold: user a preferences on music might be similar to user b and b preferences might be similar to c, but it doesn't mean a preferences are similar to c as the similarity between their preferences could have become too small after these two steps.

The research in this area is still in its early phase and results about the systems performance are mixed. For example, [29] report that overall, social-network based recommendations are no more accurate than those derived from traditional Collaborative Filtering approaches, except in special cases, such as controversial items or for cold-start situations. Others have showed that adding social network data to traditional Collaborative Filtering improves recommendation results [19].

2.4.4 Evaluation metrics

The choice of a good evaluation measure is still a matter of discussion. A large variety of metrics can be applied, and we must make a decision about the most appropriate algorithm for our scenario. It is unlikely that a single metric would outperform all others over all possible scenarios. Selecting the proper evaluation metric to use has a crucial influence on

the selection of the final recommender algorithm.

A classification of possible tasks and a proposal of suitable metrics for every tasks can be found in [12].

The next sections discus the main evaluation metrics used in recommender systems when the task is one of classification. We define POS as the set of items that the user will like (and therefore should be recommended) and NEG as the set of items that the user will not like (and therefore should not be recommended). We define as True Positives (TP) the set of items recommended that belong to POS; True Negatives (TN) is the set of items not recommended that belong to NEG; False Positives (FP) is the set of recommend items that belong to NEG (and therefore should not have been recommended); False Negatives (FN) is the set of not recommended items that belong to POS (and therefore should have been recommended).

Accuracy

Accuracy is commonly used to evaluate the performance of the recommendation method. Accuracy = (TP+TN)/(TP+TN +FP+FN)

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.17}$$

Two popular measures of accuracy for regression tasks are the Mean Absolute Error (MAE):

$$MAE(f) = \frac{1}{R_{test}} \sum_{r_{ui} \in R_{test}} |f(u,i) - r_{ui}| \tag{2.18}$$

and the Root Mean Squared Error (RMSE):

$$RMSE(f) = \sqrt{\frac{1}{R_{test}} \sum_{r_{ui} \in R_{test}} (f(u,i) - r_{ui})^2} \tag{2.19}$$

Recall and precision

For a given number of returned items, *recall* is the percentage of relevant items that were returned and *precision* is the percentage of returned items that are relevant.

$$Precision = \frac{TP}{TP + FP} \tag{2.20}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.21}$$

Note that there is an inherent tradeoff between recall and precision and must therefore be used together.

F_1-measure

F_1-measure puts recall and precision into the same measure:

$$F_1 = \frac{2RP}{R + P} \tag{2.22}$$

ROC

Receiver Operating Characteristic (ROC) Curve draws True Positive Rate ($TP/(TP+FP)$) in function of False Positive Rate ($FP/(TP+FP)$). Note that ROC curve reflects a ratio between TP and FP, and thus at the best ROC point we will obtain the best precision for this model. The advantage of ROC curves is that allows us to understand in which zones our recommender operates better. Model 1 in Figure 2.12 suggest that this model is fairly better when extracting a few items while model 2 is more consistent, outperforming model 1 for higher False Positive Rates. Coordinate (0,0) means that no item has been recommended yet, while in (1,1) all items have been recommended.

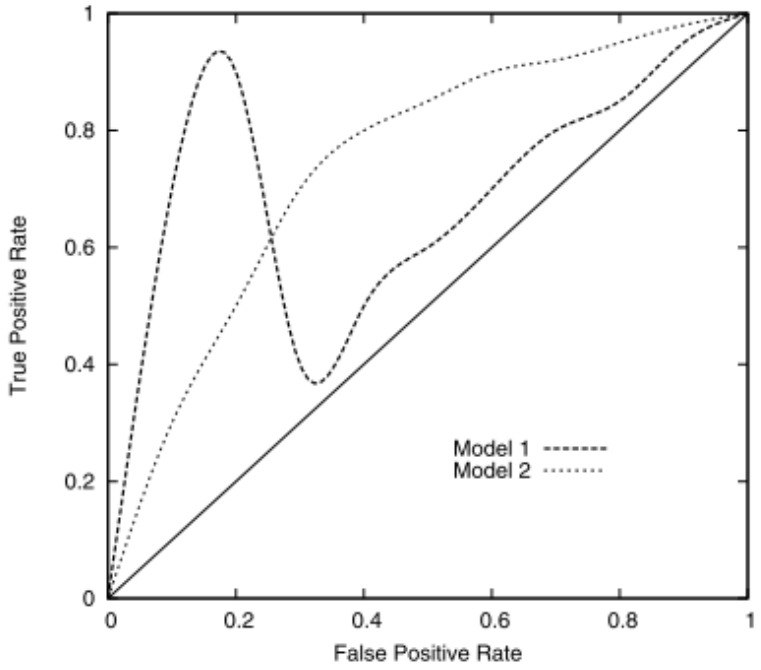

Figure 2.12: Example of two ROC curves. The diagonal represents ROC curve corresponding to a random model. The ideal curve would be one going from (0,0) to (0,1) and to (1,1), that corresponds to a model that finds all the True Positives before any False Negative.

Confidence intervals

When estimating the accuracy of a classifier by averaging several tests it is expected to get a normal distribution of our variable "accuracy". Its is intuitive to see that the higher the variance of the sample, the higher the margin of error on estimating the real mean of the accuracy. Confidence intervals are error margins under which the mean is contained with a certain probability.

Significance tests

When choosing one from a set of algorithms we must be confidant that the candidate will also perform better for the yet unseen data that the system will face in the future.There

is a possibility that the algorithm that performed best on the test set did so because the experiment was fortuitously suitable for that algorithm. To reduce the possibility of such statistical mishaps, we must perform significance tests on the results. A standard tool for significance testing is the p-value. P-value is the probability that the obtained results were due to luck. Null hypothesis (H_0) is usually defined as the two means being equal, and p-value can then be seen as the probability that H_0 is true. We will reject the Null hypothesis if the p-value of the difference is above a certain threshold. Traditionally, people choose p = 0.05 as their threshold, which indicates less than 95% confidence. Student's distribution is a probability distribution that comes out naturally from measuring the differences on the means of two populations. In machine learning tasks, we often need to compare two algorithms. From each algorithm we generate a number of tests that generate a population of results. If the mean of one population is significantly higher (e.g.: a p-value of 0.05) than that of the other, then we can confidently say that this algorithm performs better. Student's t-test looks at the average difference between the performance scores of algorithms A and B.

For a deeper discussion on what significance tests can be used what scenarios, and specially in multiple datasets, see [6].

2.5 Microblogging

Microblogs are social networks where users share short messages. These messages are mainly text, but can link to all kind of content (video, image, or web pages). Users can subscribe to other users profile to receive their messages on their main feed page. Identi.ca[1] and Twitter[2] are two of the most popular microblogging sites.

Twitter is a microblogging service where users share their messages with other users subscribed to their feed (*followers*). Every shared message is called a *tweet*, and can optionally contain links to a more detailed source of information. When users receive a tweets in their feed they can *retweet* (forward to their friends) mark it as a *favourite* (e.g.: to read it later), or *reply* to the first user about the tweet, maybe provoking a conversation on the subject of the tweet.

Twitter users have created a common markup vocabulary: RT stands for retweet, a '#' followed by a word represents an hashtag, and a '@' followed by a username directly addresses the message to that user, while keeping the message public for other users to read. This type of messages are often referred to as @replies.

Notice that a user can see other users tweets without being directly connected, via monitoring a hashtag or by receiving a retweet from one of her friends.

While initially intended for users to share what are they doing, the creativity of users and the simplicity of the service has made Twitter to accomodate a wide variety of use-cases, from political campaign to education, and from marketing to social activism.

A common use-case in Twitter is that of sharing news. Lots of users tend to specialize their profiles on one subject (e.g: computers, sports, politics) and share links to news that may become of interest to their followers. As every user only follows to a set of users he is interested on, there is a lot of potentially interesting information from people they do not follow that remains unseen to the users. Even reading everything their friends (*followed*)

[1] http://indeti.ca
[2] http://twitter.com

publish is an stressful task when the number of friends is too high. Twitter users face then a problem of information overload. That opens a door for recommender systems.

Most of recommender systems on twitter are based on people recommendation. That is, they suggest to users new people to follow. The problem is that users do not have to be interested on everything one person says, and subscribing to this user tweets, while adding some useful information to their timeline, might also increase the noise. Is therefore necessary to consider single tweets recommendations, guaranteeing that we are delivering new information with a high signal/noise ratio.

Chapter 3

State of the Art

The emerging of social networks such as Twitter, Facebook, LinkedIn, or eBay brings some new problems and opportunities to the recommender systems arena. A common problem that affects recommender systems is that of *sparsity*. Most items are only shared by a small number of users, which makes finding user similarities a very difficult task. In social networks, if we can access to the interactions between users and its relations (e.g. friendship) we can try to introduce the concept of trust to enhance traditional recommendation methods. Trust-aware recommender systems compute trust that users have on other users and include this social information in their algorithms to improve the accuracy of their predictions.

If the system where we apply recommendations is not originally designed for recommendations, it will not have any feedback mechanism. How to get (or estimate) this feedback is a problem that must be addressed. Twitter, for instance, does not have any explicit feedback on whether a user liked a friend's tweet of not. It seems clear that the accuracy of the recommender system will strongly depend on how well we extract this feedback information. Besides, in trust-aware recommender systems, even when this feedback is available there is still a need to decide how to estimate trust from this feedback.

This chapter gives an overview of the main tasks of trust-aware recommender systems and some of the most popular solutions in the literature.

3.1 Trust in social recommender systems

To understand the different strategies proposed in the literature, we propose a general classification of the different steps that are implicitly of explicitly followed in most of works on trust-based recommendations. We identified three major steps that are described in the next sections.

3.1.1 Direct trust computation

Direct trust is defined as the trust between two users that are directly connected in the social network. The first step to compute trust among users in the network consists on creating a network where nodes (users) are connected to other nodes by directional edges and where the strength of the edge corresponds to the amount of direct trust flowing through the two connected nodes.

We can distinguish between two main strategies to obtain direct trust. The first one is

explicit annotation of direct trust relationships. Golbeck [8] proposes a WOT (Web-of-Trust), an extension of FOAF (Friend-of-a-Fiend) where users annotate a set of friends they trust on in a $[0, 10]$ scale. Appleseed [30] mines Advocato[1] network where users certificate peers according to four levels of trust: namely "Observer", "Apprentice", "Journeyer", and "Master". The authors of Appleseed mapped those levels to real numbers for their computation. The advantage of these method is that they have a real (not normalized or ranked) scale of trust. In MoleTrust [18] , a similar method to TidalTrust, authors also consider information provided by users in their Web Of Trust.

The second one is *implicit inferring* of direct trust relationships. This is necessary when we do not have any user's annotation and we have to analyze users' behavior to infer direct trust relations from those users that interacted directly. Eigentrust [14] proposes analyzing the successful downloads in a peer-to-peer network to estimate how much a peer should be trusted. PageRank [21] can be seen as a global trust metric that proposes using the number of links as an indicator of trust.

3.1.2 Trust propagation

For two people who are not directly connected, the information about their mutual trust is not directly observable. However, the paths connecting them in the network contain information that can be used to infer how much they may trust one another. Paths are generally composed of edges that correspond to direct trust relations in the network. A common strategy for trust propagation is that of the random walk approach. Random walk approaches compute the probability of walking from one node to another given initial transition probabilities between some of the nodes in the network. These initial transition probabilities are stored in a transition matrix that is multiplied by itself n times to compute the probability of walking from every two nodes after n steps. PageRank [21] uses a random walk approach to compute a global trust for every website. EigenTrust [14] uses random walk to compute the global trustfulness of a peer in the network.

More sophisticated methods try to explicitly adapt the properties of trust (e.g. decay, transitivity) to their algorithms. This is clearly seen in Golbeck's TidalTrust [8]. TidalTrust incorporates two concepts that the authors detected in social networks: shorter paths are more desirable and more trusted neighbors are generally more accurate. MoleTrust [18] proposes a method similar to TidalTrust. The main novelty in MoleTrust is that, unlike TidalTrust which uses the shortest path as maximum path length, the horizon (maximum length for any propagation path) is a fixed parameter.

Another original approach is taken by Ziegler and Lausen [30]. They propose Appleseed, a trust propagation strategy based on spreading activation models to compute trust of nodes from a set of reference seed nodes. Given an amount of energy injected to every seed node, energy is proportionally distributed among the edges of the graph according to the strength (direct trust) of the edge. If a node does not receive enough energy, they say the node *runs dry*. When every node in the graph runs dry the spreading algorithm stops.

3.1.3 Trust-aware recommendations

When applying trust to recommendations, the most common approach is using trust over a Collaborative Filtering (CF). CF tries to identify users that have relevant interests and

[1] http://www.advogato.org/trust-metric.html

preferences by calculating similarities among user profiles. The idea behind this method is that consulting the behavior of other users with similar rating activity may be beneficial to one's search. To make a prediction of the rating that a user u would make of an item i, CF calculates a weighted deviation of the ratings for the item i made by u's closest neighbors and add it to the average ratings of u:

$$\hat{r}_{ui} = \overline{r}_u + \frac{\sum\limits_{v \in \mathcal{N}_{i(u)}} w_{uv}(r_{vi} - \overline{r}_v)}{\sum\limits_{v \in \mathcal{N}_{i(u)}} |w_{uv}|} \tag{3.1}$$

where $\mathcal{N}_{i(u)}$ is the set of u's closes neighbors that rated item i, r_{vi} is the rating of i made by user v, $\overline{r}_v$ is the average of v's ratings ,and w_{uv} is the similarity between users u and v. In standard CF, the set of closer neighbors $\mathcal{N}_{i(u)}$ are those users with a highest similarity to u that have rated item i. Similarity between two users' profiles is often calculated as Pearson's Correlation Coefficient (PCC).

After trust is computed and propagated, it can be introduced Equation 3.1 by means of the set $\mathcal{N}_{i(u)}$, w_{uv} or both. O'Donovan and Smith [20] call these approaches *trust-based filtering* and *trust-based weighting* respectively. In their *trust-based weighting* they combine their trust metric with Pearson's correlation, while the set of neighbors is obtained by using the former as similarity measure. In their *trust-based filtering* they use their trust metric to obtain the closest neighbors while they keep a traditional similarity measure.

Lathia et al. [17] propose k-Nearest Recommenders (kNR), instead of the traditional k-Nearest Neighbors (kNN), to find the nearest neighbors. Unlike kNN, each user's neighborhood is dynamic, and the selection of neighbors is guided by the item that a prediction is being made for.

3.2 Related work

In the following sections we will explain what we consider the most interesting approaches to trust computation and trust-aware recommender systems. Even if some of the explained methods to compute trust are not explicitly designed to be applied in recommendations - since their output is a trust estimation between users in the network-, they can be used to recommendations.

The selection has been made by relevance and proximity to our approach, which will be explained in the following chapters.

3.2.1 TidalTrust

TidalTrust [8] is an algorithm for propagating trust in networks with continuous rating systems. Direct trust is explicitly obtained by previously building a WOT (Web-of-Trust), an extension of FOAF (Friend-of-a-Fiend), where users annotate a set of friends they trust on in a $[0, 10]$ scale. Golbeck analyzed some of the properties of trust in social networks to design a trust propagation algorithm that took them into account. This observations are (a) shorter paths are more desirable than longer ones, and (b) the more trusted a neighbor is, the more coincidence in the trust ratings of the neighbor and the target user. Besides, in order to avoid fixing a maximum length of the paths where trust propagates through that would cause some interesting nodes to be unreachable, Golbeck incorporated a variable path length.

Let t_{is} represent the trust rating from node i to node s (sink). The inferred trust rating is given by Equation 3.2.

$$t_{is} = \frac{\sum_{j \in adj(i) | t_{ij} \geq max} t_{ij} t_{js}}{\sum_{j \in adj(i) | t_{ij} \geq max} t_{ij}} \tag{3.2}$$

In order to apply this formula, we need to decide which paths must be followed from the source to the sink, and which is the value of the *max* trust threshold. This threshold is dynamically computed and is used to disregard edges where trust is too low.

The full algorithm is illustrated in Algorithm 1. In the first part, the source node begins a Breadth First search for the sink. It polls each of its neighbors to obtain their rating of the sink. Each neighbor repeats this process recursively, keeping track of the current depth from the source. Besides, each neighbor keeps track of the strength of the path to it, calculated as the minimum of the source's rating of the node that leads to the neighbor and the node's rating of its neighbor. Nodes adjacent to the source will record the source's rating assigned to them. The neighbor records the maximum strength path leading to it. Once a path is found from the source to the sink, the depth is set at the maximum depth allowable. Since the search is proceeding in a Breadth First Search fashion, the first path found will be at the minimum depth. The search will continue to find any other paths at the minimum depth. Once this search is complete, the trust threshold is established by taking the maximum strength of the trust paths leading to the sink. An example of this first part of the algorithm is illustrated in Figure 3.1.

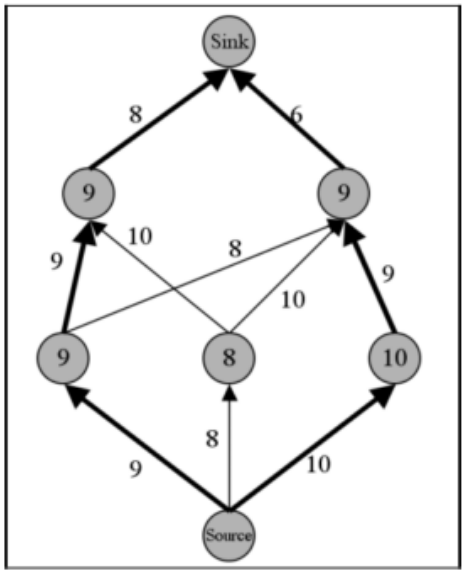

Figure 3.1: Example of trust computation with TidalTrust. The label on each edge represents the trust rating between nodes. The label on each node indicates the maximum trust strength on the path leading to that node. The two nodes adjacent to the sink have values of 9, so 9 is the *max* value. The bold edges indicate which paths will ultimately be used in the calculation because they are at or above the max threshold (source: [8]).

In the second part of the algorithm, and once the *max* value is established, a backwards

process starts from the sink to the source, through the selected paths where every node is at or above the *max* threshold, to apply Equation 3.2.

3.2.2 EigenTrust

The goal of EigenTrust [14] is to compute global reputation of peers in a peer-to-peer network. Kamvar et al. define a *local trust value* (direct trust) s_{ij} as the difference between satisfactory and unsatisfactory transactions with peer j:

$$s_{ij} = sat(i,j) - unsat(i,j) \tag{3.3}$$

To allow local trust values to be further aggregated, they define a *normalized local trust value*:

$$c_{ij} = \frac{max(s_{ij}, 0)}{\sum_j max(s_{ij}, 0)} \tag{3.4}$$

Though normalization allows aggregation, it avoids an absolute interpretation as the values are now relative. That is, if $cij = cik$, we know that peer j has the same reputation as peer k in the eyes of peer i, but we do not know if both are very reputable, or if both are mediocre.

In order to propagate trust from i to j, EigentTrust adopts a Random Surfer approach. Assume C is a matrix containing all the c_{ij} values. To propagate direct trusts up to n steps, we compute:

$$\vec{t_i} = (C^T)^n \vec{c_i} \tag{3.5}$$

where c_i is the normalized local trust vector of peer i. Since C defines a Markov Chain, if n is big enough every $\vec{t_i}$ converges to the same $\vec{t}$. This vector, which is the stationary distribution of the Markov Chain, contains the global trust of every peer in the network. Therefore, for any initial distribution $\vec{e}$ global trust can be expressed as:

$$\vec{t} = (C^T)^n \vec{e} \tag{3.6}$$

where, for simplicity's sake, $\vec{e}$ can be defined as a uniform probability distribution $e_i = 1/m$. To incorporate a priori trust in a set of P nodes, they use an initial distribution of $p_i = 1/|P|$ if $i \in P$, and $p_i = 0$ otherwise. This distribution will converge faster than the uniform probability distribution previously proposed. For inactive peers, in order to avoid undefined c_{ij} they assign $c_{ij} = 0$. EigentTrust also aims to avoid malicious collectives, that is, groups of malicious peers who know each other, who give each other high local trust values and give all other peers low local trust values in an attempt to subvert the system and gain high global trust values. EigenTrust addresses this by forcing every peer in the network to place some trust in the *a priori* trusted peers. The final algorithm is shown in Algorithm 2.

Since in a distributed peer-to-peer network there is no centralized server that stores all the values c_{ij}, Kamvar et al. propose a further modification to this algorithm to allow a distributed computation. Further details of this are out of the scope of this section and can be seen in [14].

3.2.3 Appleseed

Appleseed [31] is a trust propagation method to compute global trust from a set of local nodes (seeds) that are considered to be the authorities in a given community or network. The global

Algorithm 1: TidalTrust algorithm

```
foreach n ∈ G do
    color(n) = white
    q = empty

TidalTrust (source, sink)
    push(q, source)
    depth = 1
    maxdepth = infinity
    while d(depth) not empty and depth ≤ maxdepth do
        n = pop(d(depth))
        push(d(depth), n)
        if sink ∈ adj(source) then
            cached_rating(n, sink) = rating(n, sink)
            maxdepth = depth
            flow = min(path_flow(n), rating(n, sink))
            path_flow(sink) = max(path_flow(sink), flow)
            push(children(n), sink)
        else
            foreach n2 ∈ adj(n) do
                if color(n2) = gray then
                    color(n2) = gray
                    push(temp_q, n2)
                if n2 ∈ temp_q then
                    flow = min(path_flow(n), rating(n, n2))
                    path_flow(n2) = max(path_flow(n2), flow)
                    push(children(n), n2)
        if q empty then
            q = temp_q
            depth = depth + 1
            temp_q = empty
    max = path_flow(sink)
    depth = depth − 1
    while depth > 0 do
        while d(depth) not empty do
            n = pop(d(depth))
            foreach n2 ∈ children(n) do
                if rating(n, n2) >= max and cached_rating(n2, sink) ≥ 0 then
                    numerator = numerator + rating(n, n2) ∗ cached_rating(n2, sink)
                    denominator = denominator + rating(n, n2)
            if denominator ≥ 0 then
                cached_rating(n, sink) = numerator/denominator
            else
                cached_rating(n, sink) = −1
        depth = depth − 1
    return cached_rating(source, sink)
```

Algorithm 2: EigenTrust algorithm

$\vec{t}^0 = \vec{p}$
repeat
  $\vec{t}^{k+1} = C^T \vec{t}^{(k)}$
  $\vec{t}^{k+1} = (1-a)\vec{t}^{k+1} + a\vec{p}$
  $\delta = ||t^{(k+1)} - t^{(k)}||$
until $\delta \leq \epsilon$

trust of a given peer is the trust of this peer in the eyes of the seed nodes. Appleseed is based on the spreading activation models of psychology. Moreover, authors provide an extension to deal with distrust statements.

Given a graph G representing a network of users or nodes, continuous weights between [0,1] are assigned to edges (e.g. direct trust). Seed nodes are activated through an injection of energy e, which is then propagated to other nodes along edges according to a set of simple rules: all energy is fully divided among successor nodes with respect to their normalized local edge weight, i.e., the higher the weight of an edge is, the higher the portion of energy flows along that edge. Furthermore, the closer a node x is to the injection source s, and the more paths leading from s to x, the higher the amount of energy that flows into x. In order to eliminate endless, marginal and negligible flow, energy streaming into node x must exceed threshold T in order not to *run dry*.

Appleseed also handles trust decay and the elimination of rank sinks by incorporating a spreading factor d. Let in(x) denote the energy influx into node x. Parameter d then denotes the portion of energy $d \cdot in(x)$ that the latter node distributes among successors, while retaining $(1-d) \cdot in(x)$ for itself.

There is a common problem when trust is not propagated as an absolute value but in terms of percentage of trust given to a node. For instance, if a node a has issued only one trust statement about a neighbor p : $W(a,p) = 0.25$, and another node b assigns full trust over neighbors q, r, s, it can happen, -within some network structures-, that the final global trust assigned to p is unfairly higher than that assigned to q, r, s since these nodes had to share the energy of their parent. In order to avoid this, Appletrust incorporates the assumption that every node has full trust on the seed, adding a virtual edges to the sink $W(x,s) = 1$ from every other node. Authors argue that this solution has two more collateral effects: It indirectly gives more energy to those peers closer to the sink, and avoids dead end nodes.

The full algorithm is shown in Algorithm 3. Further details can be seen in [31]

3.2.4 kNR trust-based collaborative filtering

k-nearest recommender (kNR) Collaborative Filtering (CF) [17] is an adaptation of the more popular k-nearest recommender (kNN) Collaborative Filtering. The authors proposed this method to apply trust instead of the traditional k-Nearest Neighbors (kNN) to find the nearest neighbors. Instead of predicting ratings from "how did users that are *similar* to me rate this item?", kNR-CF predicts them from "how much do those I *trust* like this item, and how should I interpret their opinion?".

Traditional similarity methods used in Collaborative Filtering, such as Pearson Correlation Coefficient, rely on a non-empty intersection between two users' profiles in order to find a measure of similarity using co-rated items. When the number of co-rated items is low or

Algorithm 3: Appleseed algorithm

Trust $s \in V, in^0 \in R_0, d \in [0,1], T_c \in R^+$

- $in_0(s) = in^0$
- $trust_0(s) = 0$
- $i = 0$
- $V_0 = \{s\}$
- **repeat**
  - $i = i + 1$
  - $V_i = V_{i-1}$
  - $x \in V_{i-1} : in_i(x) = 0$
  - **forall the** $x \in V_{i-1}$ **do**
    - $trust_i(x) = trust_{i-1}(x) + (1 - d) \cdot in_{i-1}(x)$
    - **forall the** $(x, u) \in E$ **do**
      - **if** $u \notin V_i$ **then**
        - $V_i = V_i \cup \{u\}$
        - $trust_i(u) = 0$
        - $in_i(u) = 0$
        - $addedge(u, s)$
        - $W(u, s) = 1$
      - $w = W(x, u) / \sum_{(x,u') \in E} W(x, u')$
      - $in_i(u) = in_i(u) + d \cdot in_{i-1}(x) \cdot w$
  - $m = max_{y \in V_i}\{trust_i(y) - trust_{i-1}(y)\}$
- **until** $m \leq T_c$
- **return** $trust : \{(x, trust_i(x)) | x \in V_i\})$

zero, the similarity measure is not significant. This is a cause of what is known as *cold-start* problem. Instead, kNR follows the next strategy: When the user a enters a rating $r_{a,i}$ for an item i, the system examines all the raters for item i, and calculates how much should the target user have trusted each of these recommenders. If we were considering this problem from the perspective of a user-item rating matrix, this process would iterate over item i's column, and for each row (i.e. recommender) b make a utilitarian evaluation of the entry compared to the user's rating r:

$$value(a, b, i) = -\frac{1}{5}|r_{a,i} - rb, i| + 1 \tag{3.7}$$

The equation, which assumes a five-star rating scale, awards the highest trust to users who rated the item exactly as the user did. If the recommender b has not rated item i, a trust score of 0 is returned. If a recommender b has rated the item, the trust score will be positive, even if the recommender's rating was the complete opposite of the user's. The computed value is used to update the trust for recommender b, which is an average of the value contributed over all the n historical ratings:

$$trust(a, b, n) = \frac{\sum_{i=0}^{n} value(a, b, i)}{n} \tag{3.8}$$

The idea is to reward recommenders who can provide information, varying the reward according to the perceived quality of the information, and to downgrade recommenders that do not have any information available. Note that, unlike similarity methods, weightings between user pairs are no longer guaranteed to be symmetrical. When predicting the rating of a user a for an item i, we select the top-k neighbors $N(a, i)$ of user a who have rated the item i. Thus, unlike kNN, each user's neighborhood is dynamic, and the selection of neighbors is guided by the item that a prediction is being made for. The prediction can then be made by the traditional Collaborative Filtering formula. However, authors propose a modification of the formula that uses a *semantic distance*. Further details can be seen in [17]

3.2.5 Profile Level and Item Level trust for Collaborative Filtering

O'Donovan and Smyth [20] developed a similar solution to that of Lathia et al. They proposed two computational models of trust based on the past rating behavior of individual profiles. These models operate at a profile-level (average trust for the profile overall) and at a profile-item-level (average trust for a particular profile when it comes to making recommendations for a specific item). They name *producer* to the user contributing in the prediction and *consumer* to the target user. Trust assignments are here based on the *correctness* of recommendations of users to the target user. In order to calculate the correctness of p's recommendations, they separately perform the recommendation process by using p as c's sole recommendation partner. A rating prediction for an item i made by a producer $p(i)$ for a consumer $c(i)$ is considered to be correct if the error is below a given threshold:

$$Correct(i, p, c) \Leftrightarrow |p(i) - c(i)| < \epsilon \tag{3.9}$$

and the consumer's trust on the producer p when considering an item i is computed as:

$$T_p(i, c) = Correct(i, p, c) \tag{3.10}$$

The full set of recommendations that a given producer has been involved in $RecSet(p)$ is given by:

$$RecSet(p) = \{(c_1, i_1), ..., (c_n, i_n)\} \tag{3.11}$$

and the subset of these that are correct, $CorrectSet(p)$ is given by:

$$CorrectSet(p) = \{(c_k, i_k) \in RecSet(p) : Correct(i_k, p, c_k)\} \tag{3.12}$$

where the i values represent items and the c values are predicted ratings.

From this, authors define two basic trust metrics based on the relative number of correct recommendations that a given producer has made. The *profile-level trust*, $Trust^P$ for a producer is the percentage of correct recommendations that this producer has contributed with:

$$Trust^P(p) = \frac{|CorrectSet(p)|}{|RecSet(p)|} \tag{3.13}$$

For a more fine-grained metric, they define *item-level trust*, $Trust^I$, which measures the percentage of correct recommendations for an item i:

$$Trust^I(p, i) = \frac{|\{(c_k, i_k) \in CorrectSet(p) : i_k = i\}|}{|\{|(c_k, i_k) \in RecSet(p) : i_k = i\}} \tag{3.14}$$

Once trust is computed, authors propose two ways of incorporating trust into the traditional Collaborative Filtering formula: *trust- based weighting* and *trust-based filtering*. In *trust-based weighting*, trust is combined with the users' similarity measure to use it as the weighs in the formula:

$$c(i) = \overline{c} + \frac{\sum\limits_{p \in P(i)} (p(i) - \overline{p}) w(c, p, i))}{\sum\limits_{p \in P(i)} |w(c, p, i)|} \tag{3.15}$$

where weights are calculated as the harmonic mean of trust and similarity:

$$w(c, p, i) = \frac{2(sim(c, p))(trust^I(p, i))}{sim(c, p) + trust^I(p, i)} \tag{3.16}$$

In *trust-based filtering*, trusts is used to decide which neighbors should participate in the prediction.

$$c(i) = \overline{c} + \frac{\sum\limits_{p \in P^T(i)} (p(i) - \overline{p}) sim(c, p))}{\sum\limits_{p \in P^T(i)} |sim(c, p)|} \tag{3.17}$$

where set of trusted procucers P_i^T is:

$$P_i^T = \{p \in P(i) : Trust^I(p, i) > T\} \tag{3.18}$$

Authors also propose a combination of both methods. For a further explanation and justification of these methods see [17].

3.3 Summary

In this chapter we have seen some proposals for computing trust and for using trust in recommender systems. Unlike most of the explained methods, we cannot use annotated trust information. As we will see in the next chapter, we will have to infer direct trust from users interactions, and what we will get will not be an absolute trust value (e.g.: real values between [0,10]) but the proportion of trust that a user gives to each neighbors. This makes us disregard the use of TidalTrust. In the other hand, we will see that our items (tweets) are extremely volatile -we could say that they"disappear" after a short period of time. This is why methods based on Collaborative Filtering are not applicable either. This leaves us with Appleseed and EigenTrust to compute trust, and with content-based recommendations to build the ground of our recommendation engine. The conceptual simplicity of EigenTrust has made us to opt for designing a similar method to compute trust.

Chapter 4

A Recommender System for Twitter

4.1 Overview

The aim of this project is to build a prototype of a trust-aware recommender system for Twitter using some of the state of the art techniques and see how they perform in a such scenario. But first, we face the same problem than a search engine when analyzing the Internet: as we do not have access to the whole network, we need to build a Twitter crawler that fetches the activity of users and makes it available for our recommender.

Regarding the computation of trust in Twitter, we propose a method similar to that of EigenTrust. We adapted the EigenTrust computation of direct trust from a P2P network to the Twitter social network. As EigenTrust, we will infer direct trust analyzing the interactions between users. There are several types of interactions in Twitter and thus this translation from interactions into trust will have to be carefully analyzed. We also face a problem of how to evaluate our trust model. Evaluating trust in a social environment is a hard task with no objective measure. In order to summarize, we address here two main questions:

(a) how can we adapt existing trust metrics to Twitter?

(b) does trust information help in recommending tweets?

For the first question we will adapt EigenTrust to our scenario and evaluate it in a similar way to that of TidalTrust, as it is also a recommender system scenario. Our recommender prototype will help us to tackle the second problem. Once we have a set of candidate tweets for a given user, we will test whether adding trust information improves classical recommendation methods.

This chapter is distributed as follows: Section 4.2 explains how trust and trust propagation are adapted to Twitter. Section 4.3 briefly discusses some of the issues on temporal dynamics that should be addressed. Section 4.4 explains the crawler we designed to get users' profiles and their tweets as well as the heuristics we used to orientate the crawler. Section 4.5 proposes adapting *query expansion* to enrich the information in the tweet by adding related words, potentially improving the results of similarity metrics between tweets. Section 4.6 explains how all these modules and techniques are put together, giving a global explanation of the whole architecture of the recommender and its workflow from crawling to final recommendations.

4.2 Trust in Twitter

If Twitter had an explicit feedback mechanism, that is, the possibility for users to rate other users (or their tweets), we could create a trust metric that considers the positive feedback towards other users. The more positive feedback a user u has given over a user v, the higher confidence on the assertion that user a trusts user b. If Twitter had a FOAF-like tagging like the WOT (Web Of Trust) proposed by Golbeck [8] where users explicitly annotate trusts on other users, we could use (not considering computationally costs for this scenario) their TidalTrust algorithm to propagate trust.

But we do not have such a feedback or annotation mechanism on Twitter. Instead, we have to analyze the interactions between users to indirectly estimate how much a user trusts another one. Using the Twitter public API we can get information on this interactions. We propose using these interactions and translating them into a measure of direct measure of estimated trust.

Twitter users interact in three ways: (a) following other user tweets (b) retweeting other user's tweet (c) mentioning another user or (d) favoriting another users' tweet. The question here is how these interactions can be cast to a measure of direct trust.

- The action of a *following* b means that a wants to receive all the tweets published by b. A user can start following someone for many reasons. The user might have been recommended by Twitter or by a friend. A common case is that b has been retweeted or mentioned many times by a person that a is already following, and a eventually decided to follow b because a thought that b publications are interesting. Following is not a clear indicator of trust. Users follow users (apart from cases of personal friendship) because they think they will be interested on what they post. However, if after a period of time this profile was not as interesting as expected, users can stop following (unfollow) others. Therefore, we can not infer much information from a following b.

- The action of a *retweeting* a tweet t from user b means that a found the content of t (or the link it refers to) interesting, and a expects her friends to like the content of t as well. If a tends to retweet b a lot, it can be inferred that a trusts b at some level. It is reasonable to think that the more a retweets b, the more confident we can be that a trusts b. Our trust model is gradual, that is, we do not compute the probability of a trust relation, but its strength. Therefore we can say that the more a retweets b the stronger is her trust on b.

- The action of a *mentioning* b on a tweet t means that a wants b to read the tweet. It can be a single tweet or a whole set of crossed mentions between a and b (e.g.: a discussion). Since mentions can be used in so many ways (e.g.: expression of disagreement or notification) we can not be sure, without further analysis, whether a mention expresses trust, distrust, or none of them. However, we think mentions usually express some kind of trust relationships (people tend to relate to those people who think like them) so we will consider a mention as a positive indicator of trust.

- The action of a *favoriting* a tweet t from b may have two meanings. One possibility is that a uses favorites as a "read it later", usually a tweet with some link to external content. Another possibility is that a wants to keep this tweet because she liked it (e.g.: it was funny, or has some content that a wants to access easily in the future). Therefore,

we can say that favorites mostly express a trust relation. However, as Twitter does not provide in the API the time when a user favorites some tweet we do not to consider favorites.

Mentions and retweets express trust, but they might not express trust with the same strength. We are sure about retweets, but mentions contain more ambiguity. As we will see later, we propose weighting these interactions to fine tune the trust model.

Some Twitter users have specialized profiles on some topic (e.g.: politics or machine learning) and some others have general profiles and post about many topics. An ideal computation of trust would detect the topics of every tweet involved in a mention or a retweet, and would increase trust only on those topics. Though the problem they tackle is different, such a topic-based influence is considered for instance in [28]. For simplicity sake, we compute trust in a general way, considering that if a trusts b on a topic T, a trusts b on any topic.

We consider users trust their friends proportionally to the number of interactions. Given a user u, she shares her total trust between every user she has interacted with (mentioned or retweeted). For instance, if user a had 10 interactions shared amongst user b (3 retweets), user c (5 retweets) and user d (2 retweets), her trust t_{ab} is 0.3, trust t_{ac} is 0.5, and trust t_{ad} is 0.2. Note a user a can interact with a user b even if a does not follow b. For instance, if user b published a tweet t and another user which is followed by a retweets t, t will be published on a's timeline. As mentioned before, we will consider weighting interactions according to whether it is a mention, or retweet. It can be formally expressed by the next formula:

$$t_{ij} = \frac{wN_{ij}^{(m)} + (1-w)N_{ij}^{(rt)}}{N_i} \tag{4.1}$$

where N_{ij} is number of interactions from i to j, N_i is the number of total interactions originated by a, and w is the weight assigned to retweets in front of mentions. Our intuition is that w should have a value between 0.5 and 1. Superindexes denote mention interactions (m) or retweet interactions (rt).

Note that by assigning a limited amount of trust to share among friends, we are losing the real magnitude of trust. Instead, what we get is an ordered list of users by their assigned trust.

Trust propagation

So far we have discussed how to infer trust strength between two users (a and b) when a has directly interacted with b. However, most users have never interacted between each other. Let c be a user followed by b and not by a. Let t_{uv} be the trust of user u on user v. The goal of a trust propagation model is to compute t_{ac} from trust t_{ab} and t_{bc}.

Note that Equation 4.1 can be seen as transition probabilities of a Markov chain if t_{ij} is seen as the probability of an interaction from i to j. If the network is interpreted as a Markov chain, we can apply a *random walk* model considering trusts as probabilities and, in general, considering trust t_{ab} as the probability for a to reach b. In Markov models, the *t-step distribution* is the distribution after taking t steps from the starting distribution. It is denoted by $\Pi^t = \Pi^0 P^t$ where Π^0 denotes the initial probability distribution over states and P^t is the transition matrix P raised to the t-th power. As we are considering trust from every user, the initial probability is set to 1 for every user and P is then a vector of ones. Thus we have $\Pi^t = P^t$. In order to avoid confusion between *steps* and *trust*, we denote steps as s. If

considering trust as transition probabilities, the probability of walking from node i to node j after s steps can be expressed as:

$$t_{ij}^{(s)} = \begin{cases} t_{ij} & \text{if } s = 1 \\ \sum_k t_{ik} t_{k,j}^{(s-1)} & \text{if } s \geq 1 \end{cases} \tag{4.2}$$

where the probability of i reaching j after s steps is seen as the probability of taking a single step to some vertex k and then taking $s-1$ steps to j. Note that the model takes into account every path from i to j and with a maximum of s steps.

However, instead of having a fixed number of steps s, we want to consider that users have an horizon of trust of one step, two steps and so on up to s steps. Moreover, we want to apply a decay probability to tune the importance of closer neighbors. In a matrix form, we can express this as:

$$T^s = \frac{1}{s} \sum_{n=1}^{s} \alpha_n P^n \tag{4.3}$$

where P is the initial transition matrix with components t_{ij} and α is a decay factor which value will be discussed in Section 5.1. Our intuition is that $s = 3$ is a good choice. Inferring trust further than this would lead to a lower accuracy on trust. To avoid dead end nodes, for those users with no outcoming interactions we artificially assign an equal amount of interactions to every other user in the network.

To reduce computing cost we truncate the transition matrix Q by deleting every trust value below a threshold. We will explain this truncation later. If P has $n \times n$ dimensions, with a naive computation it takes time of $O(sn^3)$.

4.3 Temporal dynamics

Users change their preferences over time. This can be caused by a personal shift of interests or because new topics appear and old topics become obsolete. In the former case, most changes occur slowly and gradually, as are more attached to the user's personal interests. In the latter, changes occur very fast (e.g.: breaking news that the user is interested on).

Another interesting aspect is temporal dynamics on the trust relationships. Over time, Twitter users start following some new users and unfollow some of their old friends. Twitter users can do so to optimize their twitter feed with information or people they are really interested on.

Ideally, these dynamics should be considered by a recommender system. But it is not the main goal of this system to deal with this kind of dynamics. However, and as the trust model drives the crawling, which is a critical part of the system, we apply a simple decay model to capture some of the trust dynamics. Our model ages the old interactions so that the newer prevail when profiling the user. We apply a forgetting factor to old interactions of a user every time she creates a new one. New retweets imply a decay on old retweets and new mentions imply a decay on old mentions. Let $\vec{v_u}$ be the vector of interactions from user u. Let $\vec{u_a}$ a new vector with one element set to one and the rest set to zero. The n-th element corresponds to the user that received the interaction from u. The decay can then be expressed as:

$$\vec{v_u} = \lambda \vec{u} + (1 - \lambda) \vec{v_u} \tag{4.4}$$

where $\vec{b}$ is a vector with all zeros but a one in the position of user b. The more dynamic the trust is, the higher value λ should have to age old interactions. Our intuition is that trust does not change dramatically and a $\lambda = 0.1$ could be adequate.

Note that this decay is applied to the transitions matrix, not directly into the trust matrix. The former will be computed from the first.

4.4 Crawling the network

Since it is not possible to get the whole data in the Twitter network we need to crawl it using the APIs provided by Twitter. These API's allow a limited amount of requests per hour, and so it is important to have a good heuristic to guide our crawling. We use the trust information to fetch those users whom the seed users trust more at every moment. The goal of trust-oriented crawling is to fetch interesting tweets for our seed users. In the best case every tweet a seed user *retweets* will have been fetched in previous crawls. That is an impossible task as we cannot perfectly anticipate users behavior due to its continue changes. Moreover, user interests (and trusted people) can suddenly change from one day to another. Detecting these peaks is out of the scope of this thesis.

A bad trust computation would lead us to a useless crawling, as we will be collecting tweets that no one of our seed users like. In this case the recommender would not work as the good candidates would have not been collected. The crawling criteria is to fetch tweets from the top influencers of every seed user.

This trust-oriented crawling is bound to our trust-propagation model. Therefore two different models would lead to two different data and recommended items. This thesis tries to see whether a simple propagation method can be useful to enhance recommendations. Finding a better model for this scenario is out of the scope of this thesis. The details of the crawler we developed are given in Section 4.6.1.

4.5 Query expansion

Query expansion is an information retrieval techniques that expands queries with related words such as synonyms or word variants. The aim of query expansion is to add more information to the query so that the search engine finds more accurate results. We apply a query expansion technique to enrich the information on the tweet. Tweets are mini-documents limited to 140 characters lenght. As short texts do not provide sufficient word occurrences, traditional classification methods based on "bag-of-words" representation have limitations. To address this problem, we propose to use search engines to incorporate the text in the results into the bag of words of the original tweet. This solution is based on that proposed in [24]. Query expansion by internet search engines allows us to deal with different languages. That is a desirable property in our scenario, as our users and their top trusted users write mostly in Catalan, Spanish and English.

We use Bing as search engine. The reasons for choosing Bing are the facility of use of their API[1], the availability of a python library, and the possibility to send queries with no

[1] http://www.bing.com/toolbox/bingdeveloper/

limitations. We considered Google, Yahoo, and Twitter Search as well. Google was discarded for its maximum query rate limitation. Yahoo offers an search API with no limitation but as a payment service. Twitter was discarded because after running some tests we saw that its search engine attempts a full match. Results in Twitter search -even if we used only some meaningful words and not the whole tweet- were only retweets of the original one. Therefore we also discarded the possibility of using a POS tagger to identify nouns in a tweet and create a less restricting query from them.

Following [24] we use up to 200 results to expand our tweets. However, in most cases Bing does not give this amount of results.

4.6 Architecture: putting it all together

To test the MarkovTrust on Twitter we designed a system that crawls users and fetches their interactions. After a period of crawling, we will be able to see how MarkovTrusts performs and whether it can serve as a basis for recommending tweets.

The system is divided in two modules: a crawler and a recommender. The crawler updates a list of most influential neighbors for every target user and fetch their tweets. It stores these tweets for the recommender to use them as its item database. The recommender learns a model for every user from the user tweets, retweets, and mentions and later makes personalized recommendations of tweets for every target user. The next sections describe these two modules in more detail.

4.6.1 Crawler

The crawling module works in a cycle of three phases: crawling, network truncation and trust updating. The details of this cycle can be seen in Algorithm 5:

Algorithm 4: Crawling cycle

```
while True do
    S ⟵ SeedUsers() ∪ TopTrustedUsers()
    foreach s ∈ S do
        statuses ⟵ GetLastUpdates(s)
        UpdateInteracctionsMatrix(s, statuses)
    end
    TruncateInteractionsMatrix() /*remove unsignificant users*/
    UpdateTrustMatrix()
    UpdateTopTrustedUsers()
end
```

At the first cycle, the crawler has only a collection of target users that have registered in the system. These users are those who will receive recommendations from the system. They serve as seed nodes for the crawler to start looking for trusted users in the neighborhood. In the next step, the algorithm iterates through every user to get their last published tweets. Every tweet is then parsed looking for interactions (mentions and retweets) and the interaction matrix is then updated. When there are no more users left, the interaction matrix is truncated by leaving only the n-top interacted users of every target user. Using the resulting matrix as

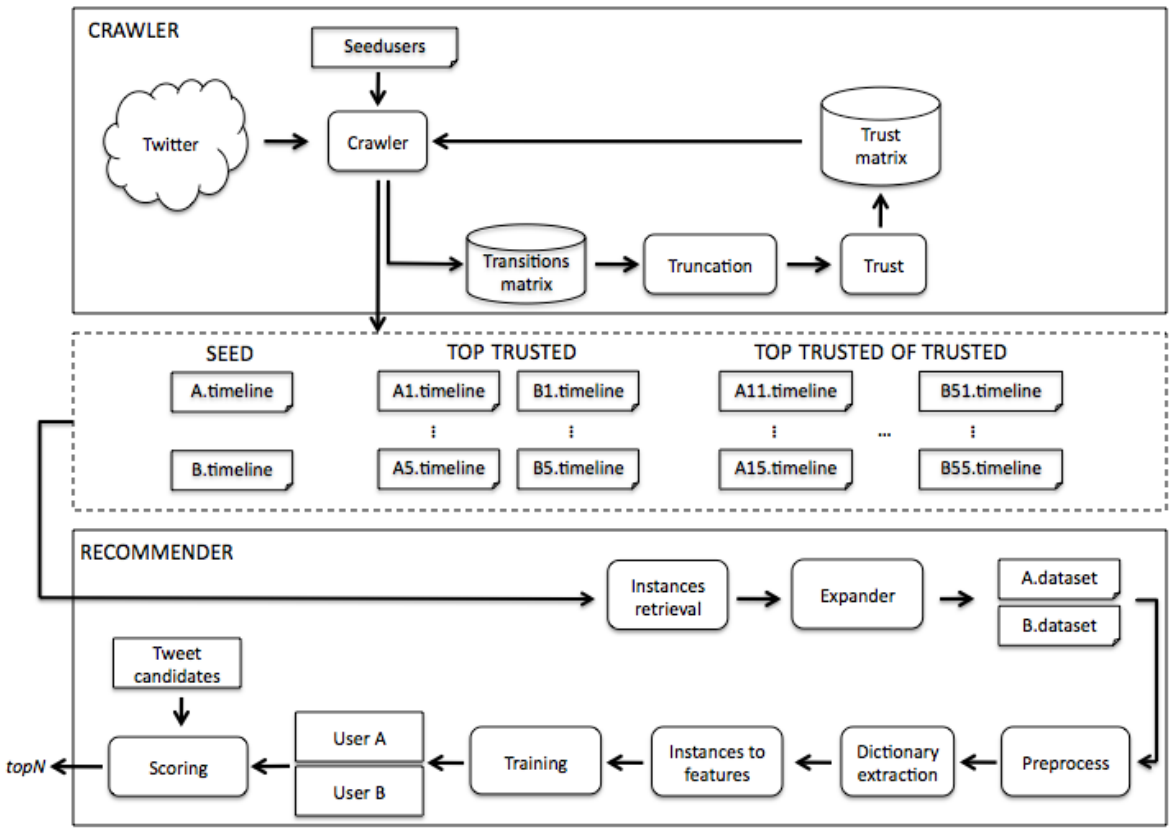

Figure 4.1: Architecture of the system considering two seed users A and B

the transition matrix, trust between every pair of remaining users is recomputed. From the final trust matrix, we get a new list of m-top trusted users that will be crawled in the next cycle. The system then sleeps for some hours in order not to overload the Twitter API.

Crawling

The aim of the crawler is to get a collection of tweets that maximizes the probability of being liked by the seed users. If trust is well computed and is significant to what users like, a good crawling would improve the possibility of making good recommendations as the items to choose from will be an interesting subset for the user. In other words, crawling should maximize the signal to noise ratio.

The selection of which paths to crawl can follow two general strategies. The first one is a similarity-based strategy. We can rank the neighbors of the seed users by how similar they are to the target seed user, and select the n-top similar. A variation of this strategy would be a more elaborated collaborative filtering strategy that recommends users that the seed user might like. However, both techniques require fetching the candidates' profiles *a priori* to create profiles and compute similarity between users. This option is discarded for the intensive (and not allowed) use of Twitter API required. A second major strategy is that of a trust-based crawling and is the one chosen for our system as it only needs to analyze the

output interactions from target users, which can be done by a single query that retrieves the last tweets of a user. New users are selected by ranking the neighbors of every seed user by how much the seed user trust them. To compute this trust we only need to parse the profile of the seed user and count the retweets and mentions to every other user. From this we can compute a trust matrix as explained in Section 4.2.

To summarize, the crawling strategy follows the next rule: For every seed user crawl her profile and the profiles of her "top trusted" users, and for every user in the "top trusted" set crawl the profiles of her "top trusted". Therefore, the system crawls the top trusted neighbors up to a distance of two steps. We consider this distance to be good enough to build an interesting set of tweets to recommend. Crawling will further increase the noise, making our recommendations achieve a possibly better recall but far less precision.

Crawled profiles are stored in separate files (one file per user) to be processed later. The information stored for every tweet is: Type, user, tweet ID, timestamp, tweet, referred user and trust. Type can be "normal" (a simple tweet with no interaction with other users), "mention", or "retweet". Referred user is an optional field that remains empty for normal tweets and for mentions and retweets, contains the user which our target user interacted with. If more than one user is contained in a tweet only the first one is considered. Trust is the computed trust of the author of the tweet on the referred user, if any. When the trust system has not computed any trust for this reference user this value is set to -1.

Pruning

As the number of iterations grows and new users are discovered the interactions matrix also grows. Computing trust relations has a cost $O(n^3)$, which is polynomial in the number of users n but costly in practice if we let n grow arbitrarily. An option is to limit n so that the size of the matrix stabilizes at some feasible size. As seen in Equation 4.3, the dimensions considered are given by the dimensions of the transition probabilities matrix P. In order to truncate P, we delete those users in the matrix with a low level of interactions. More specifically, for every user u in the network we keep the edges to the top-m trusted users and delete the rest of the edges. Once we have delete the less trusted edges, those users with no incoming edges are deleted from the matrix. We chose $m = 100$ to limit the time and memory requirements without introducing much bias in the real trust distribution.

Updating trust

After the transition matrix is updated and the less trusted users have been deleted from the matrix, the trust matrix T^s is recomputed. This matrix contains the trust index between every pair of seen users, considering that trust can be propagated up to two steps.

At every cycle the crawler reads the set of top-N trusted users for every seed user and fetches their lasts statuses. Note that, as Algorithm 5 shows, the system updates the transitions matrix for every node up to a distance of s. The new trust matrix is then computed from this transition matrix. From this trust matrix, the crawler will start a new cycle fetching the last tweets from the target users users (that are always the same), their top trusted nodes, the top trusted of the top trusted nodes, and so on up to a distance of s. This way, the crawler gets a collection of tweets from the top trusted users in the neighborhood of every target user.

Algorithm 5: Crawling cycle

```
while True do
    S ⟵ SeedUsers() ∪ TopTrustedUsers()
    foreach s ∈ S do
        statuses ⟵ GetLastUpdates(s)
        UpdateInteracctionsMatrix(s, statuses)
    end
    TruncateInteractionsMatrix()
    UpdateTrustMatrix()
    UpdateTopTrustedUsers()
end
```

4.6.2 Recommender

The aim of the recommendations module is to periodically (e.g.: every day) get a rank of top-N items for every final user. For this, it analyzes final users publications to create a user profile. Then, for every final user, it gets a list of tweet candidates to be recommended. This list is extracted from tweets published by its neighbors (top trusted nodes and their top trusted). From this list, a scoring function predicts a score for this item from the user. The highest top-N items are then showed to the user. We explain this process with more detail in the next sections.

Instance retrieval

First, a file is created for every target user with a set of positive and negative instances. As Twitter has no explicit mechanism of rating tweets, we consider a binary rating where retweets are tagged as positive. The question now is how to get or identify negative tweets, that is, tweets the user is not interested in. We follow a similar reasoning to that of [13] where, in a web search scenario: they consider that if user u clicked through result 7 but not through result 6, he must have seen 6 but not felt interested in it. Here, we consider that given a user u and a user v and given two contiguous publications $v(i-1), v(i)$ from user v, if user u retweets $v(i)$ but not $v(i-1)$ then it means that user liked $v(i)$ and not $v(i-1)$. We are making two assumptions here: first, that user u read $v(i-1)$. This might be not true if $v(i)$ and $v(i-1)$ are very separated on time. In this case, when u checks his feed he might see $v(i)$ but not $v(i-1)$. However, for simplicity's sake we assume that users can read most of their feed. The second assumption is that the first assumption holds even when u is not subscribed (follower) of user v, that is, a tweet from v has been propagated through the network until reaching the feed of user u. Actually, that is why our interaction model does not consider direct friends but any user, as possible interactions are not limited to friends (followees).

To get positive and negative instances we read user u tweets and look for retweets. For every retweet, we check the author of the original tweet. If the author was between the top-N trusted nodes at the moment of the retweet, the crawled should have captured the original tweet beforehand and maybe the previous ones. Then, we add the original tweet as positive example and the previous one as a negative example. We use the original tweet because sometimes users modify the original one to add their opinion, write it in a personal way, or

shorten some words to fit the tweet in the 140 characters allowed. If the original tweet of a retweet is not found in the logs, the example is discarded. Note that we could use Twitter API to get this tweets but this would increase the number of calls to the API.

Tweet Expander

Text similarity is a important pillar when recommending text items. Traditional methods to compute text similarity are based on bag-of-word representation of texts, to compute afterwards a similarity function such as cosine between the two vectors of words. When dealing with short texts the probability of a word coincidences falls and these methods give inaccurate results.

The aim of query expansion is to add more words to the tweet so that the word occurrences between tweets increase. We follow the work on [24] and apply a similar method to expand tweets. The idea of the method is to query a search engine with our tweet and append the words contained in the results to the tweet. First, we clean the tweet from any artifact that could tighten the search (url, hashtags, "RT", and usernames). Second, we query the Bing search API. Every result contains some fields such as URL and description. Description is the snippet of text that is shown for every result. We get the snippets of text of the first 200 results and add them to the original text of the tweet. Note that the search engine might not find 200 results for our tweet query. In fact, given the extension and complexity of most of tweets in comparison to a traditional user query, Bing finds no results or just about five or ten in the best cases. The main risk of tweet expansion is getting much more noise (unrelated results) than signal (related results).

Preprocessing

Tweets from the crawler are raw tweets from users. Users tend to use abbreviations ("u 2 are right") , hashtags (#recsys2012), url links, code words (RT, via, @user, +1), and grammatical mistakes. Then, data must be cleaned beforehand. The aim of preprocessing is normalizing the text (tokenization, filtering stop words and stemming) and cleaning it from non-text artifacts (numbers, url). As url can be informative they will be added later as a boolean feature that says whether a tweet contains a url or not.

Dictionary extraction

The aim of dictionary extraction is to create a dictionary of words that will act as word features. A common technique is to use most frequent words in the set of documents. This avoids having an excessively large set of word features and would not only slow down but make our classifier more inaccurate (the curse of dimensionality). However, as our dataset for every user is relatively small we add every word to the dictionary.

We create two dictionaries. The first one is a simple bag-of-words and the second is a Term Frequency weighted bag-of-words. In the second case, for every word in the dictionary we compute the log ratio between its occurrences in the positive examples and the occurrences in the negative examples. The formula is:

$$tf(w) = \log \frac{POS_w}{NEG_w} \tag{4.5}$$

the *tf* value is 0 when the word is equally used in positive and negative examples. It will be positive when predominant in positive examples and negative when predominant in negative examples.

Instance to features

In this step we convert the instances into sets of features so that a traditional classification algorithm can work with them. We extract the following features: *trust*, *url*, *words* which we describe below.

- *trust* is a real value [0,1] that indicates the trust from the target user to the user who made the original tweet (at the moment of the retweet).
- *url* is a binary variable that indicates whether the tweet contains any url or not.
- *words* is a set of features that expresses the content of the tweet. This information can be codified in different ways. The basic form is a boolean bag of words (one if the word is in the tweet and 0 if it is not) with words taken from the previously extracted dictionary. We can express the bag of words with the *tf* value of every word, as explained in Section 4.6.2. We can also collapse these bag of words by computing the distance from the tweet to the positive corpus.

Training and scoring

Finally the system is trained over all the past tweets and is ready to score a new tweet candidate. These candidates are taken from the top trusted neighbors. If we want the system to recommend the top-N tweets every day, candidates tweets will be those tweets in the neighborhood that have been published today (or after the last recommendation to the user was made).

Note that ten retweets talking about some new topic must be more important than one hundred tweets talking about another topic two months ago. To give some adaptivity to the system in terms of concept drift, a decay factor should be applied to past tweets. For now, we are not considering any decay factor for tweets.

4.7 Technicalities

There are somme dilemmas faced during the design of the system.

As researchers, analyzing a network such as Twitter brings some limitations. First, we do not have access to the whole network. Twitter freely provides a public API with strong limitations for developers in order not to overload their network. Using this API limits the extension and frequency of the crawling that our system can make to fetch either tweets or interactions between users. Our crawler has to fetch Twitter data without reaching the API limits. As a consequence, there is a limitation of the users of our system (target or seed users) and a limitation on how many neighbors of these users we crawl.

Recommender systems can be tested online or offline. Online approaches test the algorithms on real time, giving us a real feedback on how our system behaves out of the lab. However, online tests are not trivial to design and they are costly as any modification in the algorithms will have to be tested again. Offline tests can use past behavior of users to use

it as test set. This allows us to run the algorithms over an over again, but the results will not be as accurate. Within this approach we assume that past behavior is a good model of present behavior, which is not true specially when dealing with rapidly changing contexts such as Twitter. What was interesting one week ago might have lost its information value today. Because of time limitation, we followed this offline approach on our tests.

These factors prevent us from getting to strong conclusions over the results of our work. We will therefore limit ourselves to discuss the obtained results to get an intuition of the quality of our model and its possible enhancements.

Chapter 5

Experiments

5.1 Experiments

To test the system we selected 20 target users from Twitter. We chose users that we personally know, so that we can ask for their participation in an eventual future user study. Their interests range from politics to professional coaching. Their main language is Catalan and Spanish, while a couple of them sometimes publish tweets in English. We crawled these users and their neighborhood for six months. After processing users' logs we got an average of 314 instances per user. These instances -positive and negative cases of retweets- are balanced (around 50%-50% of retweets and non-retweets).

5.1.1 Validation of the trust model

We have no direct way of testing the accuracy of our trust propagation model. The ideal scenario would be having a full annotated network where every user has rated every other user. As we do not have such a network, we follow a similar approach to that of Golbeck [8] to see how our trust model does on direct neighbors. Later, we will test how trust information affects the accuracy of recommendations on this network. Golbeck did the following process for every user: For each neighbor n_i of the user (source), a list of common neighbors of the user and n_i was compiled. For each of those neighbors, the difference between the source rating and n_i rating of the neighbor (i.e. computed trust) was recorded as a measure of accuracy. We call this difference Δ. A smaller Δ means higher accuracy. But, unlike Golbeck, we are not interested in the absolute value of trust but on the resulting ranking of users ordered by their given trust. We would like to know whether a common neighbor is similarly (say, highly) trusted by both users, without caring for what "high" means for each user. For instance, let a be a user whose most trusted friend is b, Let b be a user whose most trusted friend is c, and let c be also a friend of a. We would like our trust model to place c between the most trusted users of a. A solution can be to compare rankings of a and b. However, this can only be done when a and b rank the same users. To solve this we normalized the rankings of our users following the next normalization formula for every item in the original ranking:

$$normalized\_rank(r) = \frac{r-1}{R-1} \tag{5.1}$$

where r is the ordinal value of the item and R is the length of the rank. The normalized ranking has all its values into [0,1].

To analyze the impact of trust decay we compare behaviors of delta when no decay is used and when an exponential and a linear decay is applied. The formula for the exponential decay used is:

$$\alpha_n = c\frac{1}{1.5^n} \tag{5.2}$$

where c is a normalizing factor to make the sum of α's to be 1. And the formula for the linear decay is:

$$\alpha_n = c(n_{max} - n + 1) \tag{5.3}$$

Table 5.1: Relation of trust rank and delta

Ranking (0-10)	Δ No decay	Δ Linear decay	Δ Exp.decay
(0-1)	1.14	0.90	0.87
(1-2)	1.10	1.22	1.09
(2-3)	0.96	1.05	1.05
(3-4)	1.07	1.18	1.20
(4-5)	1.09	0.81	1.01
(5-6)	1.13	1.11	0.92
(6-7)	0.91	1.08	1.16
(7-8)	0.90	1.08	0.99
(8-9)	1.3	1.34	1.34
(9-10)	1.26	1.11	1.16

Table 5.1 shows the average values of Δ at every position of the ranking of trust. The trust rank and Δ have been normalized to a [0-10] range. We would expect a smaller Δ in the top users, i.e., more agreement among common neighbors. However, we see no clear correlation between neighbors ratings and Δ. This might be caused either by our model being wrong or by transitivity not holding in this scenario.

If a user a trusts a user b, this can have a double meaning: First, it can reflect a general interest on what b publishes. As direct trust is computed by analyzing the interactions between users this is, by definition, a characteristic of our model; A second characteristic is that of transitivity when endorsing other users. If a trusts b and b trusts c, a would trust c if a) transitivity holds and b) the propagation model is appropriate. The general validity of our trust model will be further discussed in the next section. As for transitivity, the lack of correlation shown in the table seems to indicate that there is no apparent trust transitivity occurring in Twitter.

It would be interesting to know whether we do not see apparent transitivity on trust because trust is not transitive on Twitter, or because our trust model does not capture it appropriately. If the original interactions are transitive (a highly interacting with b and b highly interacting with c implies a highly interacting with c) we would infer that our trust model does not keep transitivity. One could even say that a model based on transitivity, such as MarkovTrust and most of trust models, might not be appropriate in such scenario.

Table 5.2 is similar to table 5.1 but uses the stored interactions instead of the computed trust. The table shows that there is a slight correlation but that it is opposite to the correlation we expected. It seems that, in terms of interactions, users agree less about those with whom

they interact more. This might be an effect of our selective crawling or due to a lack of more data to make the patterns statistically sound. Anyway, no transitivity is apparent in the interactions.

Table 5.2: Relation of interactions rank and delta

Ranking [0-10]	Δ
$[0-1)$	1.36
$[1-2)$	0.75
$[2-3)$	1.14
$[3-4)$	0.83
$[4-5)$	0.78
$[5-6)$	0.52
$[6-7)$	1.06
$[7-8)$	0.53
$[8-9)$	0.57
$[9-10)$	0.17

Even if there is no evidence of transitivity, we observe a low Δ at every position of the rank. That is, either considering interactions or inferred trust, common neighbors are similarly trusted (or interacted).

5.1.2 Influence of trust on recommendations

We would like to test whether our trust model is good enough to enhance the performance of a recommender system in Twitter. In order to test this, consider the recommender system explained in Section 4.6. Given a set of retweets a user has made (positive instances) and another set not retweeted by the user (negative instances), our recommender system tries to learn a model to predict whether a non-seen tweet will be retweeted by the user. As we are applying offline experimentations and we do not have access to future tweets, we split this collection of tweets on training set (75%) and test set (25%). As these test tweets are past tweets, we know whether the user retweeted them or not. In other words, the instances are all tagged. We compared models in a search space as shown in 5.3. We tested both SVM (RBF kernel with $C = 1.0$ and γ set to the inverse of the number of features) and Naive Bayes classifiers combined with the multiple techniques explained in Section 4.6: Trust, tweet expansion and encoding of word features. Metrics used are accuracy, recall, precision, F1, and AUC.

For intuition, Table 5.3 shows the averages of the models with trust and the ones without trust for different metrics. Though the small size of the dataset does not allow to draw rigorously sound conclusions, we observe that models using trust information seem to perform better, and in particular recall seems to be most benefited without compromising precision. However, note that we are using a class-balanced set so that the baselines are all 50%. These poor results of our recommender are probably due to the difficulty of the task at hand and the simplicity of our model. Using more sophisticated attributes would probably enhance these results.

Table 5.3: Accuracy of retweet prediction

	Classifier	Expanded	Encoding	Accuracy	Recall	Precision	F1	AUC
Trust	NB	yes	bow	50.76	59.43	52.07	55.51	50.93
			tf	52.38	58.99	52.52	55.57	52.58
		no	bow	48.79	53.01	47.76	50.25	49.88
			tf	51.97	50.49	50.60	50.44	51.51
	SVM	yes	bow	45.84	61.00	30.22	40.42	50.53
			tf	47.63	51.36	46.95	49.06	47.94
		no	bow	46.25	68.42	32.47	44.04	50.00
			tf	47.79	46.38	48.44	47.39	47.32
Averages				48.93	56.13	45.13	49.08	50.09
No trust	NB	yes	bow	47.02	57.58	48.90	52.89	49.56
			tf	51.13	49.56	54.81	52.05	52.22
		no	bow	49.28	47.98	50.76	49.33	50.54
			tf	46.12	45.18	45.84	45.51	46.28
	SVM	yes	bow	45.22	55.83	27.05	36.44	50.06
			tf	49.23	49.36	51.47	59.39	49.80
		no	bow	43.40	34.87	17.59	23.38	50.22
			tf	47.11	41.55	48.87	44.91	47.32
Averages				47.31	47.73	43.16	45.49	49.50

Chapter 6

Conclusions and future work

In this Master's Thesis we developed MarkovTrust, a model to estimate trust between users based on users interactions. MarkovTrust is not actually a novel method, but an adaptation of other well-known methods of trust propagation. We applied this trust model to Twitter and studied some of its properties. To test the utility of MarkovTrust we developed a recommender system framework that first crawls the Twitter network following the top trusted neighbors of our target users, and then makes tweet recommendations based both on past retweets and estimated trust. Though not statistically sound due to lack of data, experiments show that such a trust model does not respect the idea of transitivity of trust. However, recommendations are enhanced when using MarkovTrust which makes us think that the model has some promise for trust-aware recommender systems.

In the next sections we detail the work done in this thesis and some opened questions and future work.

6.1 Work done

These are the main contributions of this Thesis:

- *a model to infer direct trust from Twitter interactions*: We proposed a method to infer trust from Twitter users' interactions. We discussed about the nature of interactions in Twitter (mentions, retweets, favorites and follows) and defined a way to map some of these interactions into trust relations. We also implemented a method to propagate this trust based on a random walk model.

- *a recommender system prototype*: We developed a recommender system prototype to test the utility of our trust model as well as the utility of other implemented methods that participate in the recommending process (e.g.: query expansion). This prototype can be used as the basis for a more sophisticated recommender system of tweets. It is possible to extend or modify the current modules of the recommender to test other methods than the ones used in this Master's Thesis.

- *a trust-aware content-based recommender system*: Trust-aware recommender systems tend to combine trust with Collaborative Filtering methods. Collaborative Filtering assumes a set of permanent items in the system that are rated by different users. Even though most scenarios face the sparsity problem (most users only rate a small portion

of items), the sparsity in our case is even worse. The volatility of tweets makes that only a tiny portion gets retweeted by someone. Moreover, since we can only crawl a small set of users, the probability to detect two users retweeting the same tweet is also very low. This is why we disregarded the use of a collaborative filtering approach and followed a content-based one that would analyze users' tweets and retweets. To the best of our knowledge, there is no previous work on trust-aware content-based recommender systems. Trust is used here as a new feature that complements a more traditional set of features extracted from the tweet (e.g.: bag-of-words).

- *a trust-driven crawler*: In order to optimize the set of users to monitor at every moment, we developed a trust-driven crawler that uses updated trust information to focus the crawling on the top trusted neighbors.

- *benchmarking of methods*: We did benchmark experiments to compare which combinations of methods best performs in our recommender.

- *analysis of trust properties*: We explored some of the properties of trust and interactions in Twitter and concluded, for instance, that there is no evidence of trust transitivity, since our experiments do not give strong evidence for it in Twitter, but rather point in the opposite. Transitivity is an important assumption in most trust propagation methods. If this is confirmed by experimentation within a larger dataset, trust propagation methods for Twitter -and maybe in other microblogging social networks- should consider this particularity.

6.2 Future work

The work done in this Thesis has also opened a set of questions that could be explored in the future:

- *study of microblogging network properties and users behavior*: An exhaustive study of the statistics and patters of this kind of social networks could help to improve -or even change- our trust metric. An first step in this path could be to analyze the marginal contribution of retweets and mentions to trust accuracy. This can be done be repeating the experiments using different weights in Equation 4.1. This study could also help to best tune the parameters chose in our crawler (i.e.: number of top trusted users to crawl according to how trust is distributed), and in our trust propagation method (i.e.: trust decay and trust horizon).

- *temporal dynamics*: We stated in this thesis that some kind of temporal decay should be applied, as tweets and interactions become obsolete when some time has passed. We applied a decay factor to interactions, but we did not apply any decay factor to old tweets.

- *tweets ranking*: Although, during our experiments, we implemented a basic ranker of tweets by an SVM-rank to get an output of tweets ordered by the probability of being retweeted by the target user, we could not test the accuracy of the ranking since it needs to be done on a online test framework. If the online framework is developed, this final module can be improved and other ranking methods can be proposed.

- *topic detection*: In this Master's Thesis we considered trust to be non-topic dependent. However, its well known that trust is actually topic dependent. In order to take this into account, a topic detection system should be developed for tweets. Different trusts would then be computed in parallel, one for each topic.

- *query expansion*: There is still room for a further study of query expansion for tweets, refining the proposed method or proposing novel ones such as adding synonyms of words. Probably, using state of the art techniques of Natural Language Processing can help in this issue.

- *advanced text processing*: Since we work with text items, the utilization of Natural Language Processing techniques can be helpful. Part-of-speech tagging and parsing can be used to extract verbs and nouns from tweets in order to use only these entities to improve the results of query expansion based on search engines. Computing the semantic distance between tweets can be done by Short Text Semantic Similarity (STSS) methods. A basic model of Latent Semantic Analysis was implemented for the experiments, but it did not give results worth mentioning. We think the number of available tweets was not enough for LSA to improve the results of a simple bag-of-words representation. Experiments with LSA in a bigger dataset should be repeated.

- *online testing*: For simplicity's sake and lack of time, we used offline experiments to test our system. However, recommender systems need to be tested online to have a more precise information of the accuracy of their recommendations. An online testing framework should be developed to get more significant conclusions. An intermediate solution would be to make a user study under a controlled environment.

Bibliography

[1] Chris Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More.* Hyperion, 2006.

[2] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[3] JS Breese and D Heckerman. Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52, August 1998.

[4] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[5] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, September 1967.

[6] Janez Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[7] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2):131–163, 1997.

[8] Jennifer Golbeck. *Computing and applying trust in web-based social networks.* PhD thesis, University of Maryland at College Park, 2005.

[9] Jennifer Golbeck. Personalizing applications through integration of inferred trust values in semantic web-based social networks. *W8: Semantic Network Analysis*, page 15, 2008.

[10] G. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5):403–420, April 1970.

[11] Miha Grčar. kNN Versus SVM in the Collaborative Filtering Framework. *Data Science and Classification*, pages 251–260, 2006.

[12] Asela Gunawardana. A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. *Journal of Machine Learning Research*, 10:2935–2962, 2009.

[13] Thorsten Joachims. Optimizing Search Engines using Clickthrough Data. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[14] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the twelfth international conference on World Wide Web - WWW '03*, page 640, New York, New York, USA, 2003. ACM Press.

[15] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. An Introduction to Latent Semantic Analysis. *Discourse Processes*, (25):259–284, 1998.

[16] T.K. Landauer and S.T. Dumais. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2):211, 1997.

[17] Neal Lathia, Stephen Hailes, and Licia Capra. Trust-based collaborative filtering. *Joint iTrust and PST Conferences on Privacy Trust Management and Security IFIPTM Trondheim Norway*, 263:119–134, 2008.

[18] Paolo Massa. Trust metrics on controversial users: balancing between tyranny of the majority and echo chambers. *International Journal on Semantic Web and*, pages 1–21, 2007.

[19] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 17–24. ACM, 2007.

[20] John O'Donovan and Barry Smyth. Trust in recommender systems. *Proceedings of the 10th international conference on Intelligent user interfaces IUI 05*, 15:167, 2005.

[21] L Page, S Brin, and R Motwani. The PageRank citation ranking: Bringing order to the web. *Technical report, Stanford*, 1998.

[22] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Handbook, Recommender Systems*. Springer, 2011.

[23] J. J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The Smart retrieval system - experiments in automatic document processing*, pages 313–323. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[24] Mehran Sahami and Timothy D Heilman. A web-based kernel function for measuring the similarity of short text snippets. *Proceedings of the 15th international conference on World Wide Web WWW 06*, pages:377, 2006.

[25] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[26] B. Schwartz. *The paradox of choice: Why more is less*. Harper Perennial, 2005.

[27] Rashmi Sinha and Kirsten Swearingen. Comparing recommendations made by online systems and friends. In *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, volume 106, 2001.

[28] Xiaodan Song, Belle L. Tseng, Ching-Yung Lin, and Ming-Ting Sun. Personalized recommendation driven by information flow. *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '06*, page 509, 2006.

[29] Patricia Victor, Chris Cornelis, and Ankur M Teredesai. Trust- and Distrust-Based Recommendations for Controversial Reviews. *Information Systems Frontiers*, (161), 2009.

[30] Cai-Nicolas Ziegler and Georg Lausen. Spreading activation models for trust propagation. *Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04)*, 2004.

[31] Cai-Nicolas Ziegler and Georg Lausen. Propagation Models for Trust and Distrust in Social Networks. *Information Systems Frontiers*, 7(4-5):337–358, December 2005.