Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Χρονικά Μεταβαλλόμενα Μέτρα Κεντρικότητας και Πρόγνωση Δεσμών σε Κοινωνικά Δίκτυα** **Time-Aware Network Centrality measure and Link Prediction**
Ονοματεπώνυμο Φοιτητή	**ΦΟΥΝΤΟΥΚΙΔΗΣ ΠΑΝΑΓΙΩΤΗΣ**
Πατρώνυμο	**ΓΕΩΡΓΙΟΣ**
Αριθμός Μητρώου	**ΜΠΠΛ16024**
Επιβλέπων	**ΔΙΟΝΥΣΙΟΣ ΣΩΤΗΡΟΠΟΥΛΟΣ, ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ**

Ημερομηνία Παράδοσης **Σεπτέμβριος 2020**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Τσιχριντζής Γεώργιος Καθηγητής	Αλέπης Ευθύμιος Αναπληρωτής Καθηγητής	Σωτηρόπουλος Διονύσιος Επίκουρος Καθηγητής

Table of Contents

ABSTRACT

In network analysis predicting future links is an important task that provides further understanding and knowledge about the evolution of a social network. As social networks are constantly increasing the link prediction has attracted much attention. In this paper, we split the social network in subsets based on the timestamp of each interaction. For each subset the similarity matrices are calculated and using the Support Vectors Machine algorithm we train the estimator. We present the results of the classification method based on accuracy of its prediction between two consecutive subsets.

ΠΕΡΙΛΗΨΗ

Στην ανάλυση κοινωνικών δικτύων η πρόβλεψη δημιουργίας νέων ακμών (συνδέσεων) προσφέρει μια καλύτερη κατανόηση και γνώση για την εξέλιξη ενός κοινωνικού δικτύου. Δεδομένου ότι τα κοινωνικά δίκτυα συνεχώς μεγαλώνουν, η πρόβλεψη ακμών έχει κερδίσει το ενδιαφέρον. Στην εργασία αυτή, χωρίζουμε το αρχικό κοινωνικό δίκτυο σε πολλαπλά μικρότερα υποδίκτυα με γνώμονα την χρονική στιγμή που δημιουργήθηκε η κάθε σύνδεση. Για κάθε ένα υποδίκτυο, τα μέτρα ομοιότητας υπολογίζονται και χρησιμοποιούνται για την εκπαίδευση του αλγόριθμου Support Vectro Machine. Τα αποτελέσματα της μεθόδου παρουσιάζονται με βάση την ακρίβεια του εκτιμητή μεταξύ δυο διαδοχικών υποδικτύων.

1 Introduction

Nowadays people are connecting through numerous social applications creating groups of people that interact with each other. In order to study these interactions and the properties of these groups or communities, the social network can be used.

Interactions and communities can be modeled as a graph, where the nodes represent the people composing this group and the edges represent the interaction between the corresponding persons. Although, one must not think this modeling as a static one, since social networks are changing dynamically over time. That means, new edges and vertices are inserted to the graph as time progress. In order to understand the way social network is changing and evolving, many variables has to be considered and it is thought to be a very complex problem. Although, a more easy problem to understand and solve is the analysis of the association between two specific nodes. In this paper we will deal with the likelihood of a future association between two nodes, that at the present are not associated with each other. This problem is the Link Prediction problem.

In order to define more accurately the link prediction problem we present below formula. Given a social network graph $(G(V, E)$ in which an edge $e = (u, v) \in E$ represents some form of interaction between u, v at a time $t(e)$. For time $t < t'$ we assume that $G[t, t']$ is the subgraph of G restricted to the edges with timestamps between t, t'. In a supervised training model we can use successively time intervals for training and testing. In this concept, problem is modeled as to find the list of edges that were not present in G[t0,t0`] but are predicted to appear in $G[t_1, t_1']$.

In this dissertation, we try to predict the creation of new links in the social network of stack-overflow. The data provided, is the links between two users, and the timestamp their interaction took place. The interactions are of three types, user a answered user's b question at time t, user a commented on user's b question at time t, user a commented on user's b answer on time t. In our study, we used all the interaction and treated them as same. In order to create the training and testing sets, we split the edge set into subsets based on their timestamps. We calculated the similarity matrices of the created nodes and then we applied the SVM algorithm to train our model.

1.1 The Link Prediction Problem

Social networks are a popular way to model interactions among people in a group or community. They can be represented as graphs, where a vertex represents a person and an edge represents some form of association between the corresponding persons. One main attribute of the social networks is their dynamic change over time. New edges and vertices are added to the graph as time passes. Understanding the dynamics that drive the evolution of a social network is a complex problem due to the large number of variable parameters.

Although, a more easy problem is to study and understand the linkage between two specific nodes. For example, some interesting questions are: How does the association patterns transform over time? What are the variables that affects the associations? How is the association between two nodes affected by other nodes? In this paper we deal with the problem of predicting the likelihood of a future association between two nodes, given that there is no association in the current state of the graph. This problem is commonly known as the link prediction problem. In effect, the link prediction problem asks: to what extent can the evolution of a social network be modeled using features intrinsic to the network topology itself? Can the current state of the network be used to predict future links?

The link prediction problem is also related to the problem of inferring missing links from an observed network in a number of domains. On a static model of network interactions, one tries to find the not directly visible links, based on the existing visible links This problem is slightly different from ours, as the graph is a static snapshot of a network while in our case the graph evolves over time. Also, tends to take into consideration specific attributes of the nodes in the network, rather than evaluating the power of prediction methods based purely on the graph structure.

For instance, Facebook's friend finder method , they could either suggest people whom you might find interesting enough that the connection may lead to actual friendship in real life or they could suggest friends that you already know but you are not connected yet. The first example refers to the link prediction problem while the latter refers to the inferring missing links problem.

Link prediction has many other applications outside of social networks. Fox example, bioinformatics, link prediction can be used to find interactions between proteins; in ecommerce implements features as the recommendation system for future purchase for Amazon; in the security domain it can help identify hidden groups of terrorists or criminals. Furthermore, many studies have been made on co-authorship networks. Two authors who are "close" in the network will have colleagues in common and will travel in similar circles; this social proximity suggesting that it is possible to collaborate in the future. So, link prediction in this application can accelerate a mutually beneficial professional or academic connection that would have taken longer to form.

2 Problem Definition

Given an unweighted, undirected graph $G = \langle V, E \rangle$ representing the topological structure of a social network in which each edge $e = \langle u, v \rangle \in E$ represents an interaction between u, v that took place at a particular time $t(e)$. For two timestamps $t \text{ and } t' > t$, let $G[t, t']$ denote the subgraph of G consisting of all edges with timestamp between $t \text{ and } t'$. Let $t_0 < t'_0 < t_1 < t'_1$. Then the link prediction task is: given the network $G[t_0, t'_0]$; output a list of edges not present in $G[t_0, t'_0]$ that are predicted to appear in the network $G[t_1, t'_1]$. The graph for time period $[t_0, t'_0]$ is the training set while the graph in time period $[t_1, t'_1]$ is the testing set.

To generate the list of predicted nodes, heuristic algorithms are used which assign a similarity matrix S whose real entry S_{xy} is the score between x and y. This score represents the measure of similarity between nodes x and y. For each pair of nodes $x, y \in V$ generally $S_{xy} = S_{yx}$.

In order to test the algorithm's accuracy, we use consecutive subgraphs the $G[t_0, t'_0]$ as the training set and $G[t_1, t'_1]$ as testing set.

For a social network G(V,E), there are V × V – E possible edges to choose from, if we were picking a random edge to predict for our existing social network. If G is dense, then E ≈ V2 - b where b is some constant between 1 and V. Thus, we have a constant number of edges to choose from, and O(1/c) probability of choosing correctly at random. If G is sparse, then E ≈ V. Thus, we have a V2 edges to choose from, and O(1/V2) probability of choosing correctly at random. Unfortunately social networks are sparse, so picking at random is a terrible idea.

In the DBLP dataset, in the year 2000, the ratio of actual and possible link is as low as 2 × 10−5. So, in a uniformly sampled dataset with one million training instances, we can expect only 20 positive instances. Even worse, the ratio between the number of positive links and the number of possible links also slowly decreases over time, since the negative links grow quadratically whereas positive links grow only linearly with a new node.

For a period of 10 years, from 1995 to 2004 the number of authors in DBLP increased from 22 thousand to 286 thousand, meaning the possible collaborations increased by a factor of 169, whereas the actual collaborations increased by only a factor of 21

3 Resolution Methods

Below are presented a brief description of commonly used prediction methods that use topology information about networks in the prediction process. Common similarity measures are common neighbors, graph distance, Jaccard's Coefficient, Addamic / Addar, Preferential attachment Katz, Hitting Time, Rooted PageRank. A short description of these measures is provided below. The measures applied in the social network link prediciton problem solution will be analyzed further in later chapters.

- Graph Distance

  Perhaps the most direct metric for quantifying how close two nodes are is the graph distance. It is defined as the negative of the shortest-path distance from x to y.

  Note that it is inefficient to apply Dijkstra's algorithm to compute shortest path distance from x to y when G has millions of nodes. Instead, we exploit the small-world property of the social network and apply expanded ring search to compute the shortest path distance from x to y.

  Specifically, we initialize S = {x} and D = {y}. In each step we either expand set S to include its members' neighbors (i.e., S = S ∪ {v|⟨ u, v⟩ ∈ E ∧ u ∈ S}) or expand set D to include its members' inverse neighbors (i.e., D = D ∪ {u|⟨ u, v⟩ ∈ E ∧ v ∈ D}). We stop whenever S ∩ D != ∅ . The number of steps taken so far gives the shortest path distance. For efficiency, we always expand the smaller set between S and D in each step.

- Common Neighbors

  The common-neighbors predictor captures the notion that two strangers who have a common friend may be introduced by that friend. This introduction has the effect of "closing a triangle" in the graph and feels like a common mechanism in real life. Newman has computed this quantity in the context of collaboration networks, verifying a positive correlation between the number of common neighbors of x and y at time t, and the probability that x and y will collaborate at some time after t.

- Jaccard's Coefficient

The Jaccard coefficient—a similarity metric that is commonly used in information retrieval— measures the probability that both x and y have a feature f, for a randomly selected feature f that either x or y has. If we take “features” here to be neighbors, then this measure captures the intuitively appealing notion that the proportion of the coauthors of x who have also worked with y (and vice versa) is a good measure of the similarity of x and y.

- Adamic/Adar (Frequency-Weighted Common Neighbors)

  This measure refines the simple counting of common features by weighting rarer features more heavily. The Adamic/Adar predictor formalizes the intuitive notion that rare features are more telling; documents that share the phrase “for example” are probably less similar than documents that share the phrase “clustering coefficient.”

  If “triangle closing” is a frequent mechanism by which new edges form in a social network, then for x and y to be introduced by a common friend z, person z will have to choose to introduce the pair ⟨ x,y⟩ from (choose |Γ(z)| with 2) pairs of his friends; thus an unpopular person (someone with not a lot of friends) may be more likely to introduce a particular pair of his friends to each other.

- Preferential Attachment

  One well-known concept in social networks is that users with many friends tend to create more connections in the future. This is due to the fact that in some social networks, like in finance, the rich get richer. We estimate how ”rich” our two vertices are by calculating the multiplication between the number of friends (|Γ(x)|) or followers each vertex has. It may be noted that the similarity index does not require any node neighbor information; therefore, this similarity index has the lowest computational complexity.

- Katz (Exponentially Damped Path Counts)

  This heuristic defines a measure that directly sums over collection of paths, exponentially damped by length to count short paths more heavily. The Katz-measure is a variant of the shortest-path measure. The idea behind the Katz-measure is that the more paths there are between two vertices and the shorter these paths are, the stronger the connection.

- Hitting Time

  A random walk starts at a node x and iteratively moves to a neighbor of x chosen uniformly at random. The hitting time Hx,y from x to y is the expected number of steps required for a random walk starting at x to reach y.

  One difficulty with hitting time as a measure of proximity is that $H_{x,y}$ is quite small whenever y is a node with a large stationary probability πy, regardless of the identity of x (That is, for a node y at which the random walk spends a considerable amount of time in the limit, the random walk will soon arrive at y, almost no matter where it starts. Thus the predictions made based upon $H_{x,y}$ tend to include only a few distinct nodes y) To counter-balance this phenomenon, we also consider normalized versions of the hitting and commute times, by defining $\mathrm{score}(x,y) =- H_{x,y}\cdot \pi_y$

- Rooted PageRank

  Another difficulty with using the measures based on hitting time and commute time is their sensitive dependence to parts of the graph far away from x and y, even when x and y are connected by very short paths. A way of counteracting this difficulty is to allow the random walk from x to y to periodically “reset,” returning to x with a fixed probability α at each step; in this way, distant parts of the graph will almost never be explored.

  Random resets form the basis of the PageRank measure for web pages, and we can adapt it for link prediction. Similar approaches have been considered for personalized PageRank, in which one wishes to rank web pages based both on overall “importance” (the core of PageRank) and relevance to a particular topic or individual, by biasing the random resets towards topically relevant or bookmarked pages.

Given all the above similarity matrices, they can be used separately or they can be combined into a matrix. A combination of all these similarity measures will increase the complexity of the problem. Although, using an approach like this, will provide a better estimation for the likelihood of two nodes to be connected in the future. In our project we have followed the approach to combine a part of these metrics. A combination similarity matrix was created for each possible edge, with columns the distinct similarity measures Graph Distance, Common Neighbors, Jaccard's Coefficient, Adamic / Adar and Preferential Attachment. In order to classify the pair of nodes as connected or not we have used the Support Vector Machine training algorithm.

4 Support Vector Machines

4.1 Introduction to Support Vector Machines

Support Vector Machine (or "SVM") is a supervised machine learning algorithm which can be used for both classification and regression problems. The objective of Support Vector Machine algorithm is to find a hyperplane in a N-dimensional space that distinctly classifies the data points. To separate the two classes of data points, there any many possible hyperplanes that could be chosen. The objective is to find a plane that has the maximum margin between the data points of both classes. Maximizing the margin distance provides more confidence that the future data points will be classified correctly.

Hyperplanes are a boundary to distinguish two classes. Data points in either side of a hyperplane can be classified to the specific class. The dimension of hyperplane depends on the number of attributes of the data points. For example, the data points with 2 attributes can be separated by a linear hyperplane.

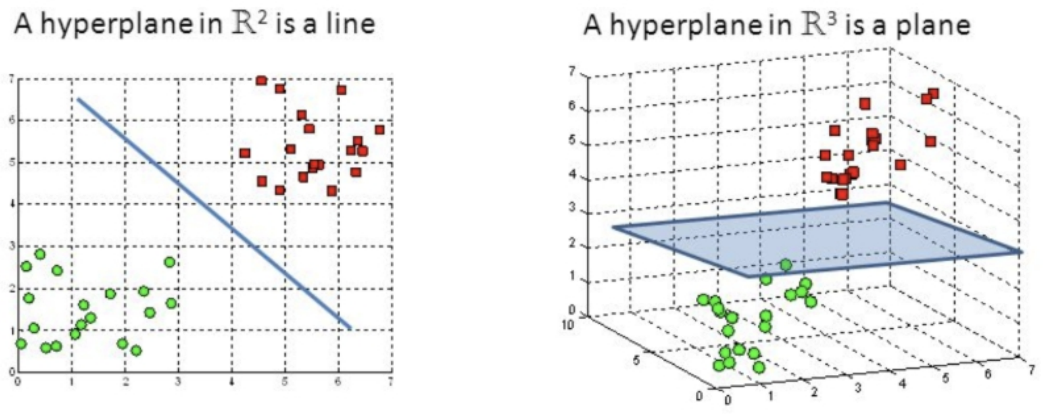


Figure 1

Support vectors are the data points that are closer to hyperplane and affects the position and the orientation of the hyperplane. These support vectors help us maximize the margin of the classifier.

Lets suppose that our dataset is not separated in two distinct classes and the hyperplane is not easily calculated, as it is shown in below figure.

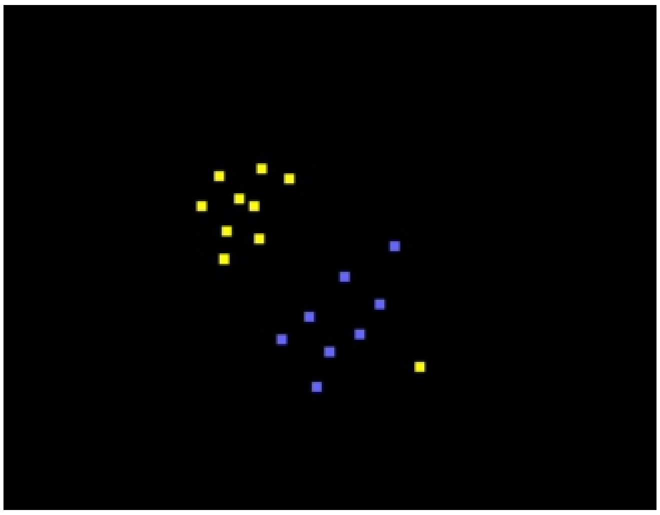


Figure 2

One yellow dot is an outlier for yellow class. SVM has a feature to ignore outliers and find the hyperplane that has the maximum margin as depicted below (figure 3). So, SVM is robust to outliers.

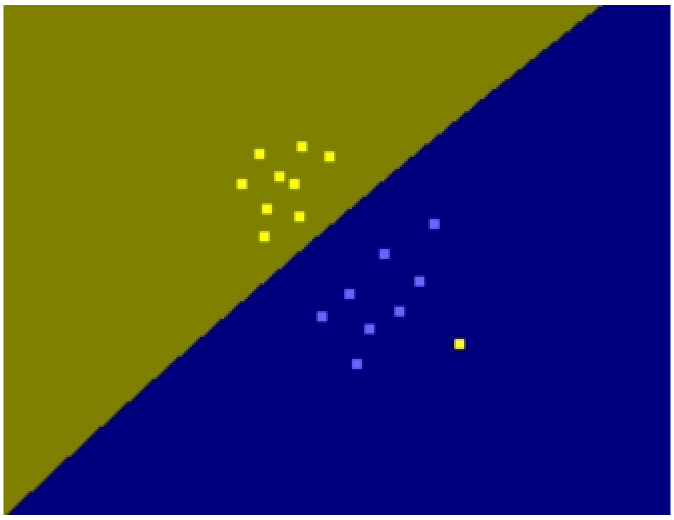


Figure 3

Another case is that we cannot have a linear hyperplane to separate the two classes as in figure 4

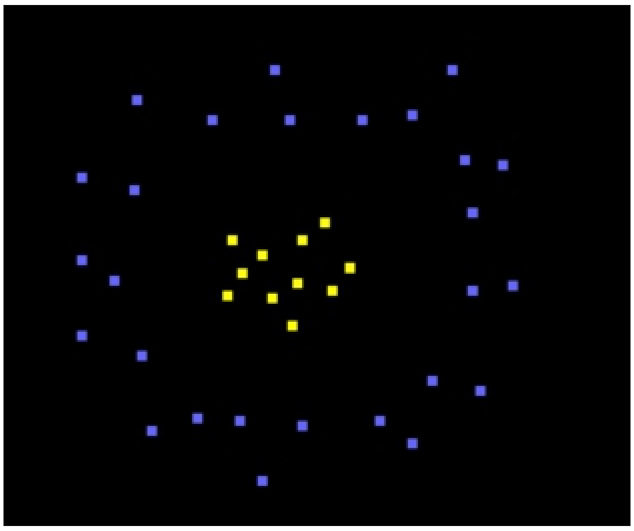


Figure 4

In this case, SVM is introducing one new variable and change the feature space from two dimensions to three. This technique is known as the kernel trick. Kernel functions take as input low dimensional space and transforms it into higher dimensional space. For support vector machine, this mean that it transforms the non separable problem into a separable problem. Some kernel function examples are the below:

- Polynomial Kernel: Function $k(x_i, x_j) = (x_i \cdot x_j + 1)^d$ where d is the degree of the polynomial
- Gaussian Kernel: $k(x, y) = \exp(-\frac{\|x-y\|^2}{2\sigma^2})$
- Gaussian Radial Basis Function (RBF): $k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$
- Laplace RBF Kernel: $k(x, y) = \exp(-\frac{\|x-y\|}{\sigma})$
- Hyperbolic tangent kernel: $k(x, y) = \tanh(\kappa x \cdot y + c)$
- Sigmoid Kernel: $k(x, y) = \tanh(ax^T y + c)$

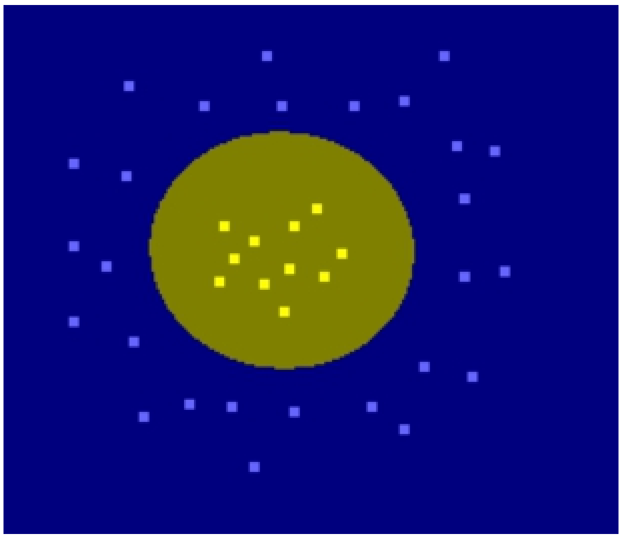


Figure 5

In our example SVM produce a circle in the 2-D space.

As a conclusion, Support Vector Machine algorithm can provide a decision function for a classification problem, just by examine a small part of the training samples. This becomes a Quadratic programming problem that is easy to solve.

4.1.1 Theoretical Background

Suppose we have a set of training points in the 2-D space like Figure 6. The problem can be modeled as:

Find a.b.c, such that

$$\mathrm{ax} + \mathrm{by} \geq \mathrm{c} \text{ for red points}$$

$$\mathrm{ax} + \mathrm{by} \leq \mathrm{c} \text{ for blue points}$$

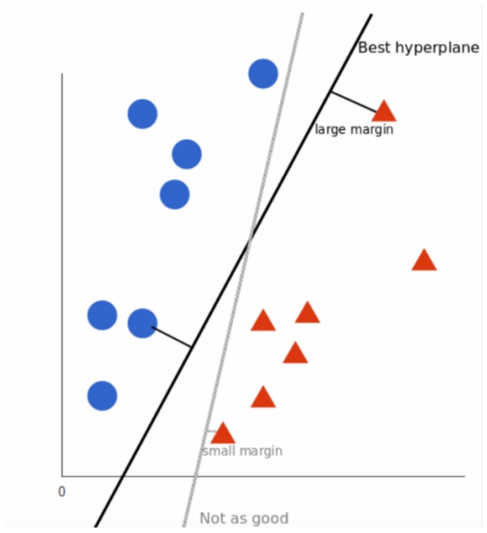


Figure 6

This problem has infinite solutions, but we are looking for the optional solution. Support vector are the critical elements that affect the position of hyperplane. These are the vectors that touch the margin.

$$w_0^T x + b_0 = 1 \text{ or } w_0^T x + b_0 =- 1$$

So now the problem can be described as:

Define the hyperplanes H such that:

$$w \cdot x_i + b \geq 1 \text{ when } y_i =+ 1$$

$$w \cdot x_i + b \leq 1 \text{ when } y_i =- 1$$

$$H_1 \text{ and } H_2 \text{ are the planes:}$$

The points on the planes of H_1 and H_2 are the tips of the support vectors.

$$H_1\colon\ w \cdot x_i + b =+ 1$$

$$H_2\colon\ w \cdot x_i + b =- 1$$

The plane H_0 is the median in between, where $w \cdot x_i + b = 0$.

d+ = the shortest distance to the closest positive point

d- = the shortest distance to the closest negative point.

The margin (gutter) of a separating hyperplane is d+ + d–.

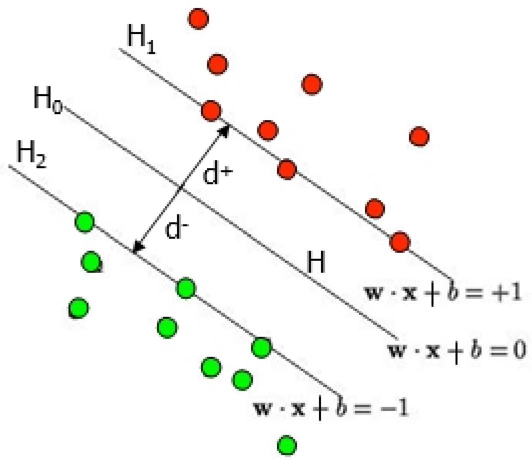

H_1
H_0
H_2
d+
d-
H
$\mathbf{w} \cdot \mathbf{x} + b = +1$
$\mathbf{w} \cdot \mathbf{x} + b = 0$
$\mathbf{w} \cdot \mathbf{x} + b = -1$

Figure 7

4.2 Pros and cons

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation

5 The Application Domain

In this chapter a description of the dataset will be provided.

In order to address the Social Network Analysis problem, the sx-stackoverflow dataset was used. The set was retrieved by snap.stanford.edu and its structure is represented below.

Dataset was created by the social interactions that took place in a stack exchange site of mass usage, stack overflow. The dataset consists of 3 columns, (source target timestamp):

- Source is the user that started the interaction
- Target is the user that received an interaction
- Timestamp is the time that an interaction took place

There are three types of interaction included in the dataset:

- User a answered user's b question at time t
- User a commented on user's b question at time t
- User a commented on user's b answer at time t

Source and target values are of type Integer, while timestamp values are of type unix timestamp.

Brief statistical view of dataset is presented in below table.

Dataset statistics	
Nodes	2601977
Temporal edges	63497050
Edges in static graph	36233450
Time span	2774 days

Table 1

One quality element of the dataset is that each edge represented in the structure of (u,v,t) is a directed edge from user u to user v. For the purposes of our experiment, we have considered the graph created at each timeslot to be an undirected graph with an interaction to be created once an edge appears. Also, an interaction between two nodes can happen for multiple timestamps inside a specific timeslot. The number of interactions between two users in a specific timeslot have not been taken into consideration, as it is of our interest only the fact that two nodes of the graph are connected. For example, if nodes a,b interact with each other more than 1 time, only the first appearance of the edge will be considered in our Social Network graph produced by the sx-stackoverflow dataset. In this way the graph created at a specific timeslot represents a slice of the constantly evolving stackoverflow community.

Also, we should point out that the node count in each graph is increasing with high rate. This is due to the fact that as time progress new users (nodes) are added to the existing ones. So, in early timestamps nodes of the graph is minimal, while in the latest timestamps the node count is huge. This add complexity to the calculation of metrics for these graphs. For example, the number of users the first day of stackoverflow is less than 50, while the last day is over 20000. Here we should note that the count of nodes is not increasing constantly, but many ups and downs are observer between sequential timeslots.Below you may find a presentation of the number of graphs and how they develop in the first 1500 timeslots.

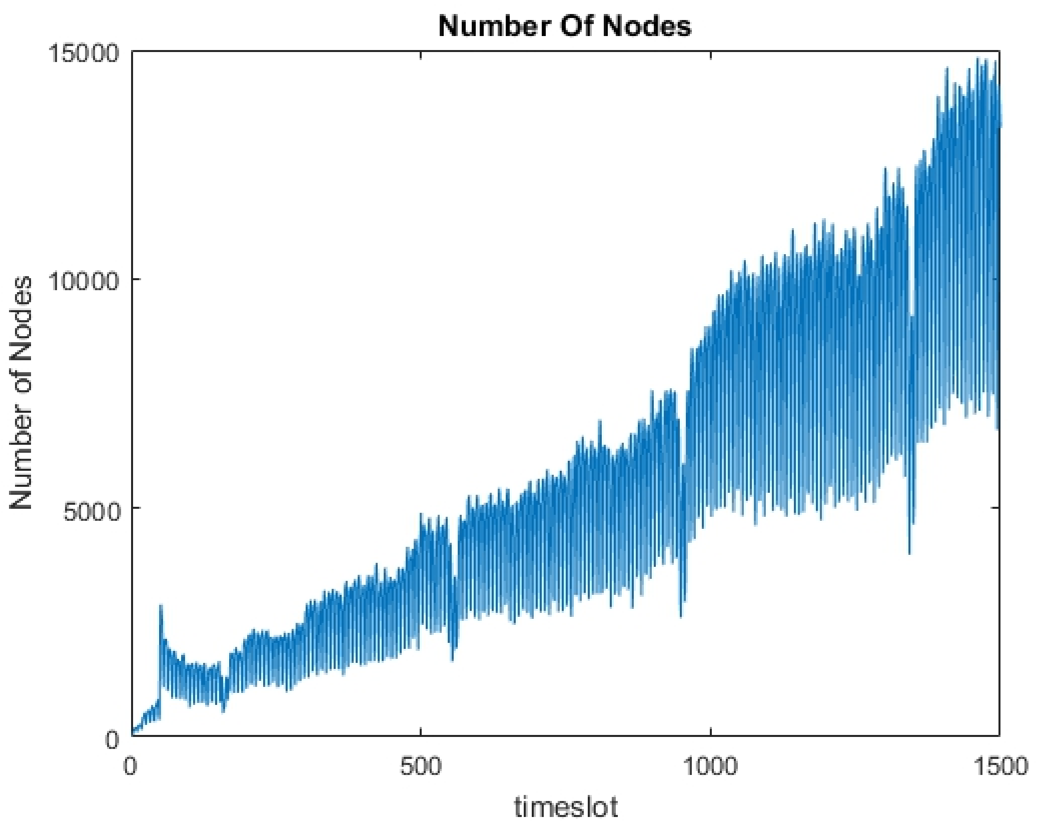


Figure 1

The biggest challenge for sx-stackoverflow dataset is the imbalance between the two classes of the set, the connected nodes and the not connected nodes. In general, the graphs generated for each timeslot are producing very sparse adjacency matrices. In a given Graph G={V,E} the number of possible edges is N*(N-1). While the edge count remains too little for these graphs. Given the fact that the problem has two classes, the first class is the existing edges and the second class is the not connected edges, the sum of the two classes is the number of possible edges in each graph. So the percentage of the connected nodes is very small in comparison to the not connected nodes class. This is creating a very biased dataset, which is creating issues with the training of the classifier. Here the most important question is, how will this classifier not classify every input to not connected class. For example, a neural network algorithm would have failed to be trained to predict which nodes will be connected in the next timeslot

An exercise where we have calculated the ratio between the number of connected nodes to all possible connections. Below a statistical analysis for these ratio per graph is presented.

Minimum	0.0131
Maximum	2.7926
Mean	0.0790
Variance	0.0263

Table 1

Below you may find a graph depicting how the evolution of the percentage of connected nodes through the first 1500 timeslots

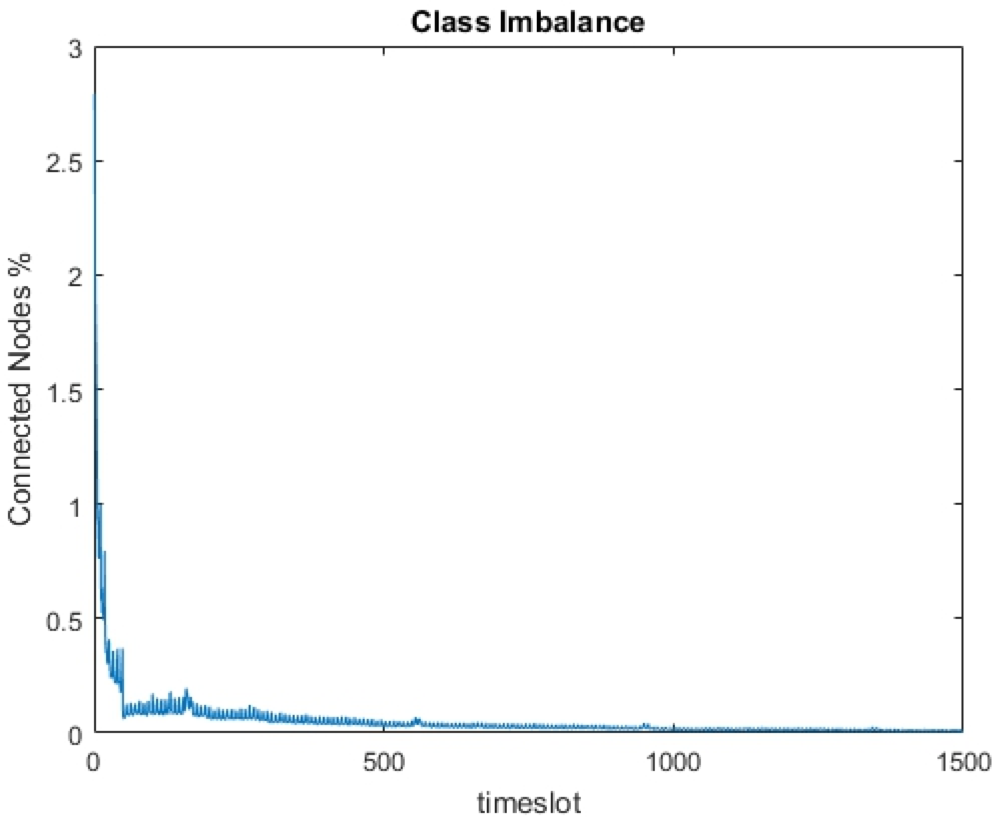
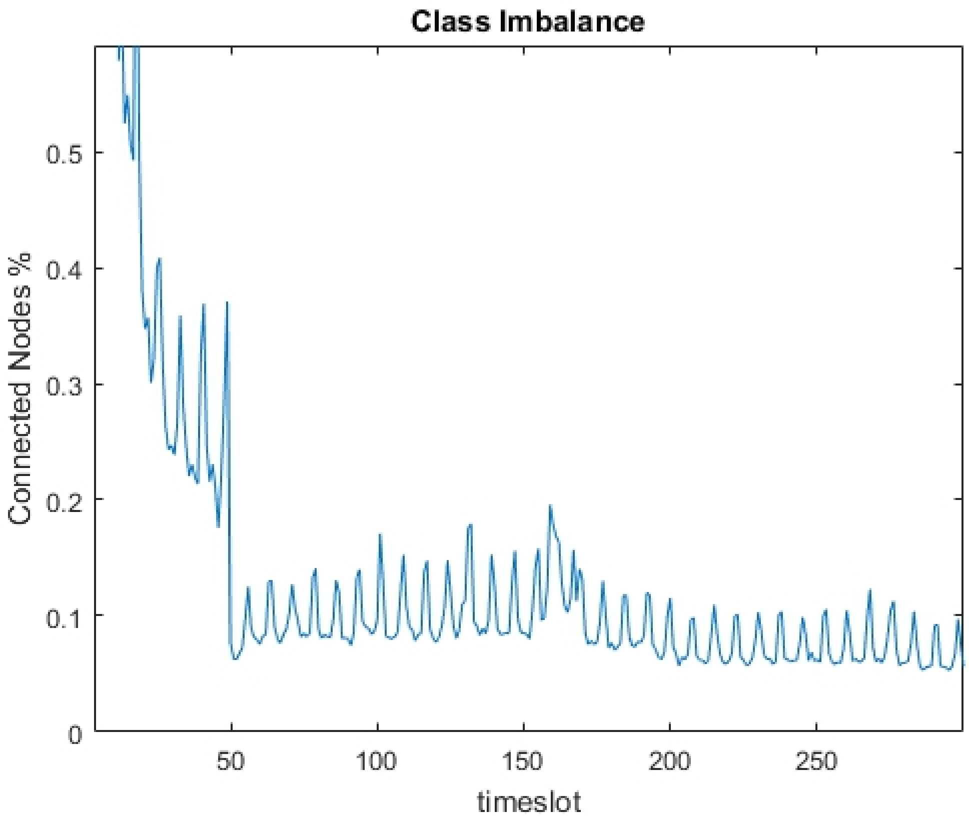


Figure 2

Also, below a graph depicting the same thing for the first 250 timeslots.



6 Classification Approach

In this chapter we will describe the creation process of the dataset division into training and testing periods, the calculation of the metrics measures and the training process of the classifier.

6.1 Dataset Division

The given sx-stackoverflow dataset is composed of triplets of the form (source, target, timestamp). The oldest timestamp is denoted as $t_{\min}$ and the latest timestamp is $t_{\max}$. Our goal is to create a sequence of N non-overlapping time periods $\{T_1,...,T_N\}$ with equal duration δt covering the entire time interval. In order to do this we can define N+1 time instances $\{t_0,...,t_N\}$ where $t_j = t_{\min} + j \cdot \delta t$, $0 \le j \le N$ where $\delta t = \frac{\Delta T}{N}$ and $\Delta T = t_{\max} - t_{\min}$. In that way we can define the j-th time period as below

$$T_j = \begin{matrix} [t_{j-1}, t_j), 1 \le j \le N-1 \\ [t_{j-1}, t_j, j] = N \end{matrix}$$

In order to achieve this breakdown I have used the Matlab tall tables functionality in order to cover the big size of sx-stackoverflow dataset. Below you may find the matlab code that was used to do the actual dataset division into N time periods. Based on the choice of N, we can have the prediction of the new connected nodes from day to day or week to week. For example if N is set as 390 this can give us an estimation for the new connection from week to week.

Although, this N choice creates very big time periods, we have chosen N to be 3000, this value creates time periods of roughly 21 hours.

```
clear all
ds=datastore('sx-stackoverflow.txt');
t=tall(ds);
%t=t(1:100000,:);
Edgeset=gather(t);
min_Ts=min(Edgeset.Var3);
max_Ts=max(Edgeset.Var3);

N=3000;

dt=(max_Ts-min_Ts)/N;
p=zeros(1,N)';

for i=0:N-1
    p(i+1)=min_Ts+i*dt;
end

for i=1:N-1
    UBidx=Edgeset.Var3<p(i+1) & Edgeset.Var3>=p(i);

    E=Edgeset(UBidx,1:2);

    filename="edgeset"+i;
    save(filename, 'E');
end

UBidx=Edgeset.Var3<=p(N) & Edgeset.Var3>=p(N-1);
E=Edgeset(UBidx,1:2);
save("edgeset3000",'E');
```

Table 2

We firstly create a datastore to import the sc-stackoverflow dataset that we load it in a tall table. Second step, we load the table in memory. Then we find the max and min values of column three of dataset. This column contains the information of timestamp, so now we have the oldest and latest timestamp, $t_{\min}$, $t_{\max}$. Then we calculate the δt. We calculate all time instances and we save them in vector p. At this step the creation of time period is completed and we need to create the subgraphs for each time period.

Each time period is described above as T_j, with $1 \le j \le N$, we may consider the corresponding undirected subgraphs of the network, denoted as:

$G[t_{j-1}, t_j] = (V[t_{j-1}, t_j], E[t_{j-1}, t_j])$ where $E[t_{j-1}, t_j] = \{e_{ij}(t) : t \in T_j\}$

We firstly found the indexes of the triplets (source, target, timestmap) where timestamp is between t_{j-1} and t_j for $1 \le j \le N-1$. Then we create a table that contains the columns that denote source and target (timestamp is obsolete for the rest of implementation, and its removed) based on the indexes of the previous step. For N time period we do the same actions outside of the for loop. As of now, for each consecutive time instances, we have a table that has every edge appeared between these instances. We save each of these tables with value edgeset_J in order to be available in later stages. Below you may find the code for creating the N subgraphs.

```
for k=1:N


    clearvars -except k


    lodFilename="edgeset_"+k;

    Edges=load(lodFilename);

    Edges=Edges.E;

    Edges=unique(Edges,'rows');







    Source=Edges.Var1;

    Target=Edges.Var2;

    V=union(Source, Target);


    Vvalues=num2cell(1:size(V));

    map=containers.Map(num2cell(V'),Vvalues);


    Em1=values(map, num2cell(Source'));
    Em2=values(map, num2cell(Target'));


    D=digraph(cell2mat(Em1),cell2mat(Em2));

    A=adjacency(D);


    A(logical(eye(size(A))))=0;

    A=A+A';

    A(A>1)=1;

    A=full(A);

    G=graph(A, 'upper');


    S = conncomp( G, 'OutputForm' , 'cell' );


    L=cellfun(@length,S);

    [M,I]=max(L);

    H=subgraph(G,S{1,I});


    A=adjacency(H);

    A=full(A);

end
```

Table 3

For each set of edges we load the corresponding table. First thing is to remove the duplicate rows, as this denotes that two users interact more than one time and as described earlier we need only the connection between two users. Then we create the set of Nodes of the subgraph, taking the union of the two columns (source and target). Union function takes into consideration the unique values of the concatenated columns. Here we should remind that source and target column consist of user ids, which mean that the values are not sequential and the min and max values of the ids can be very sparse. So if we created the graph at this point, using as input the id values, that would create a huge sparse graph (even more sparse than the nature of this problem). So we have created a mapping that map each id to the set $\{1,2,..N\}$ where N is the number of ids existing. In this way we created a more concise graph. Then we create the directed subgraph with input the values of source and target tables. In order to create an undirected graph do a transformation using the adjacency matrix that is created from the digraph. Firstly we zero the main diagonal of the adjacency matrix. From the directed graph we add itself with its transpose and for each value greater than 1 we assign the value 1. And now we have an undirected graph G for each time period. During our analysis, we saw that there where many nodes that were connected only to one node. In order to lower the complexity of the calculation of the similarity matrices, we have used only the biggest connected subgraph of each graph. In this way, we removed a large number of possible connected edges, with sacrificing a small number of connected nodes.

At this point, the creation of graph of each time period is completed. To sum up, we have removed all duplicate edge appearance and we have created an undirected graph for the calculation of the similarity matrices.

6.2 Similarity Matrices Calculation

The core element of the link prediction problem is the similarity matrices. These values will be the input of the SVM algorithm either for training or for testing purposes. The similarity matrices calculated are graph distance, common neighbor, Jaccard's coefficient, Adamic / Adar and preferential attachment.

At this step we have created the graph for each time period. Now for every possible edge we will calculate the above mentioned similarity matrices. At first, we calculate the graph distance using the distance function of Matlab graph library and then we multiply the outcome with -1 in order to change the sign of the values provided. For the pair of nodes that have graph distance 1, that means that are directly connected, we need to update this value to zero. In case the value remains as 1, this would be a strong indicator to the classifier that this two nodes are connected. This would have as a result, the outcome of the classification not to be accurate. In order to calculate this value we create a temporal graph that this edge is removed and for this graph we calculate the geodesic distance.

For the calculation of common neighbors, all we do is a matrix multiplication. As per graph theory, if A is the adjacency matrix of a given graph G, the matrix $B = A^k$ where $k \geq 2$ provides the k-length walks between two nodes. For k=2, this provide the matrix of walks with length 2, which denotes that the two nodes have a path of the form u->x->v. In that notion x is the common neighbour of u,v. So, the $B = A^2$ matrix has at each cell $[i, j]$ the number of common neighbors between the nodes i, j .

The next similarity measure is the Jacard's Coefficient. For each pair of nodes we use the calculated in previous step number of common neighbors. Then we find the neighbors of each node using the neighbors function of Matlab graph library. Then we take the union of the two arrays and count the number of neighbors using the numel function of Matlab. The final value of Jacard's Coefficient for each pair of nodes derives from the division of the number of common neighbors and the number of neighbors.

For Adamic / Adar similarity measure we iterate through the common neighbors of two nodes. Common neighbors array is calculated using the neighbors function of Matlab graph library. For each common neighbor of the two nodes, we calculate the number of neighbors of this node. Then the logarithm is applied on this value and we sum up the reverse fraction.

The last measure calculated is Preferential attachment. For each pair of nodes we calculate the number of neighbor for each node using neighbors function and numel function. Then we multiply the number of neighbors of each node.

As a last step, we update all the infinite values to zero. For the training of the SVM we need only the values of the upper triangle of the similarity matrices. This is due to the fact that our graph is undirected. The Matlab code that was used for the calculation of the above similarity matrices is presented in below table. The variables name Sxx denotes a measure:

- Sgd --> geodesic distance
- Scn --> Common Neighbors
- Saa --> Adamic / Adar
- Sjc --> Jaccard's coefficient
- Spa --> Preferential attachment

```
Sgd=distances(H);

Scn=A*A;

Saa=zeros(size(A));

Sjc=zeros(size(A));

Spa=zeros(size(A));

Scn2=zeros(size(A));


for i=1:size(A,1)

    for j=i+1:size(A,1)

        ni=neighbors(H,i);

        nj=neighbors(H,j);

        CommonNeighbors=intersect(ni,nj);

        Scn2(i,j)=numel(CommonNeighbors);

        Spa(i,j)=numel(ni)*numel(nj);

        Sjc(i,j)=Scn(i,j)/numel(union(ni,nj));


        ob=numel(CommonNeighbors);

        for z = 1:ob
Saa(i,j)=Saa(i,j)+1/log(numel(neighbors(H,CommonNeighbors(z))));

        end

        if Sgd(i,j)==1

            tmp=A(i,j);

            A(i,j)=0;

            Gtemp=graph(A, 'upper');

            TempDist=distances(Gtemp);

            Sgd(i,j)=TempDist(i,j);

            A(i,j)=tmp;

        end

    end

end

Sgd=-1*Sgd;

Saa(isinf(Saa))=0;

Sjc(isinf(Sjc))=0;
```

```
    Spa(isinf(Spa))=0;

    Sgd(isinf(Sgd))=0;

    Scn(isinf(Scn))=0;


    A(A==0)=-1;

    idx=logical(triu(ones(size(A))));

    vec=[Sgd(idx) Saa(idx) Scn(idx) Sjc(idx) Spa(idx) A(idx)];
```

Table 4

6.3 SVM Training

As described above, we have calculated all the above mentioned similarity matrices and we have saved them as the set $\{S_{GD}, S_{AA}, S_{CN}, S_{JC}, S_{PA} \mid A\}$ where A is the value denoting if the two nodes are connected or not, in other words the class of each set. The matrices were calculated for all successive time intervals. So, let M_i to be the set of $\{S_{GD}, S_{AA}, S_{CN}, S_{JC}, S_{PA} \mid A\}$ at time interval i, M_i is used as the training set for the Support Vector Machine algorithm. At first we create a matrix , X, of the similarity matrices from the training set M_i containing only the first 5 columns, which will be the input for our classifier. Then, we create the target vector, Y, which contains the last column of M_i. For the training of the SVM classifier the fitcsvm Matlab library was used. A detailed documentation for this function can be found in Matlab fitcsm. So, at this point we have created a variable to hold the information of each SVM classifier, "model", X is the predictor matrix while Y is the class label vector. For input argument 'KernelFunction', we ran a number of tests to determine the more optimal function which was the polynomial function of order 2. Additionally, we have chosen to scale the predictor matrix values, as we enabled the 'Standardize' option that is available for the fitcsvm. This option centre and scales the predictor matrix attributes based on the weighted mean and standard deviation.Also name value pair arguments for logging reasons have been used to provide a more clear view of the training of the SVM classifier. The related code is depicted below.

```
%training the SVM for a given set of matrices.

clc

RESULTS=zeros(99,4);

Train_RESULTS=zeros(99,4);

for k2=1:99

    clearvars -except k2 RESULTS Train_RESULTS

    k3=k2+1;
    lodFilename1="similarityMatrices_tail_2_"+k2;

    lodFilename2="similarityMatrices_tail_2_"+k3;


    vec1=load(lodFilename1);

    vec2=load(lodFilename2);


    vec1=vec1.vec;

    vec2=vec2.vec;


    Xtest=vec2(:,1:5);

    Ytest=vec2(:,6:6);


    Ytest(~any(Xtest,2),:)=[];

    Xtest(~any(Xtest,2),:)=[];


    X=vec1(:,1:5);

    Y=vec1(:,6:6);


    Y(~any(X,2),:)=[];

    X(~any(X,2),:)=[];



    model = fitcsvm(X ,
Y ,'KernelFunction','polynomial','PolynomialOrder',2,'KernelScale','au
to','Standardize',true,'Verbose',1,'Prior','empirical');

    sv=model.SupportVectors;


    train_label = predict(model, X);

    C = confusionmat(Y,train_label);



    label = predict(model, Xtest);
    C2 = confusionmat(Ytest,label);







    RESULTS(k2,1) = C2(1,1);
    RESULTS(k2,2) = C2(2,1);
    RESULTS(k2,3) = C2(2,2);
    RESULTS(k2,4) = C2(1,2);

    Train_RESULTS(k2,1) = C(1,1);
    Train_RESULTS(k2,2) = C(2,1);
    Train_RESULTS(k2,3) = C(2,2);
    Train_RESULTS(k2,4) = C(1,2);

end
filename="Results_3";
save(filename, 'RESULTS');

filename="Train_Results_3";
save(filename, 'Train_RESULTS');
```

Table 5: SVM Training

7 Classification Results

For each consecutive time periods, T_i, T_{i+1}, the similarity matrix of T_i is used to train the SVM model and similarity matrix of T_{i+1} is used as a testing set. So, the model that is created in training period is used as a predictor for the testing set and the output values of the model is compared to the actual values of the testing set. In order to evaluate the results, four values are used, true positive, false positive, true negative and false negative $[TP, FP, TN, FN]$. If the predicted value of a pair of nodes is 1 (that means they are connected) and the actual value for this set is also 1 then we have predicted an actual connection and this is denoted as true positive. In case the predicted value of the pair of nodes is 1 and it is not actually connected then we have predicted wrong that this edge exist, so we denote these cases as false positive. The same logic applies for the pair of nodes that the prediction result is not connected and the actual value of this pair is not connected, then we have true negative case. If the prediction of the SVM for a pair of nodes is not connected and the actual value is connected then this is a case of false negative. At this point we should highlight that there is a difficult in the training of the classifier as the actual not connected nodes are the dominant element of the network in hand, as we have explained in previous chapters. This high percentage can bias the classifier because if it classifies all the pair of nodes as not connected the accuracy of the classifier will be over 90%.

The below figures presents the result for the testing accuracy of the SVM training. For lower computational complexity we present the results for the training of the last 100 time periods. Also, the training and testing was performed on the biggest neighborhood of each time period.

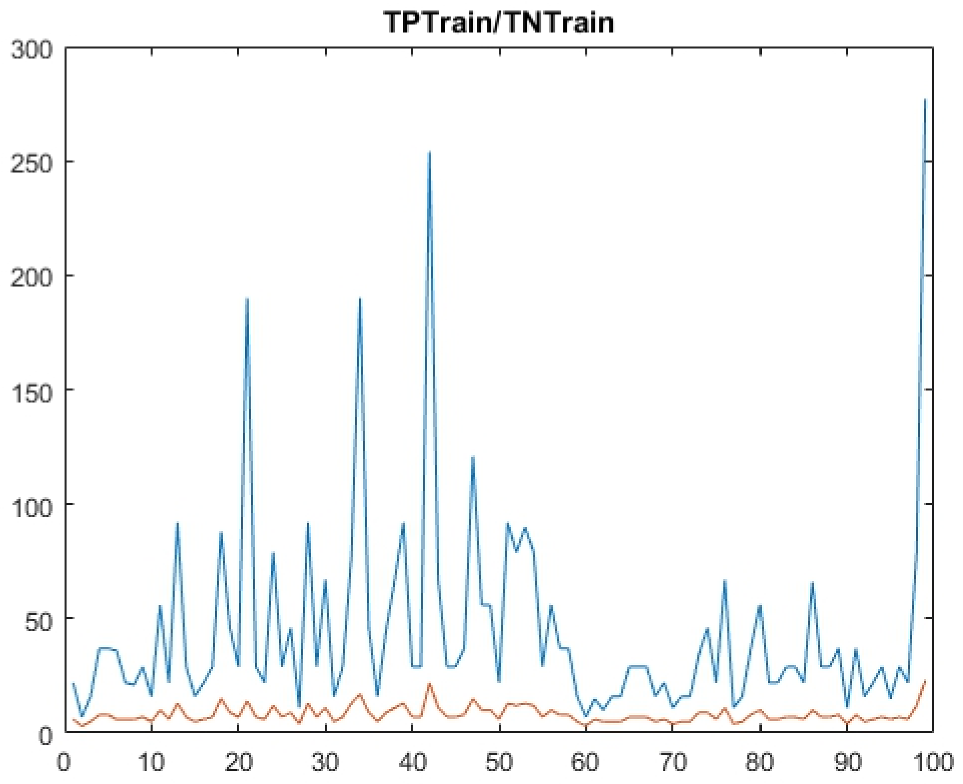


Figure 3: TPTrain to TNTrain results

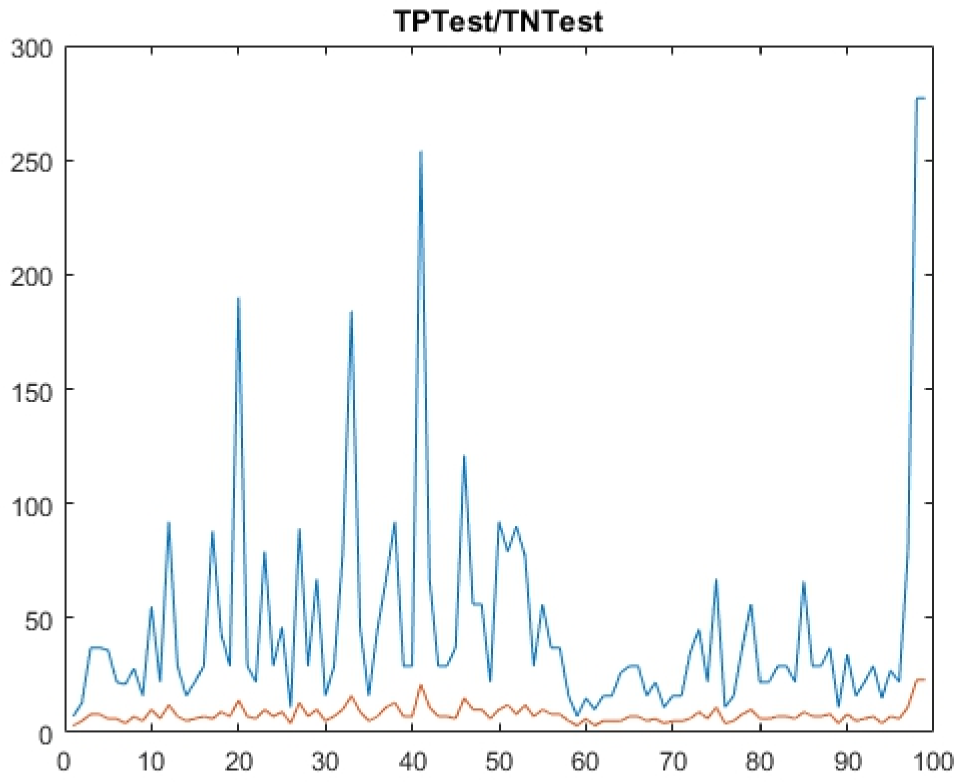


Figure 4:TPTest to TNTest

The SVM classifier applied to this data works great for small neighbors, its accuracy in predicting the newly created edges is 100% in most of the experiments. Although, in case we apply the same classifier in the entire graph, this classifier offers no accuracy as it classifies every pair of edge as not connected. The same applies for the not connected edges that are predicted correctly from the SVM. For small graphs, as when it is applied on the biggest neighborhood, the true negative results accuracy is 100% for most of the experiments. Of course, when the SVN is applied in the whole graph the true negative accuracy percentage is above 96% for all cases based on the graph properties that were presented in above chapters. The performance of the SVM is highlighted in below figures.

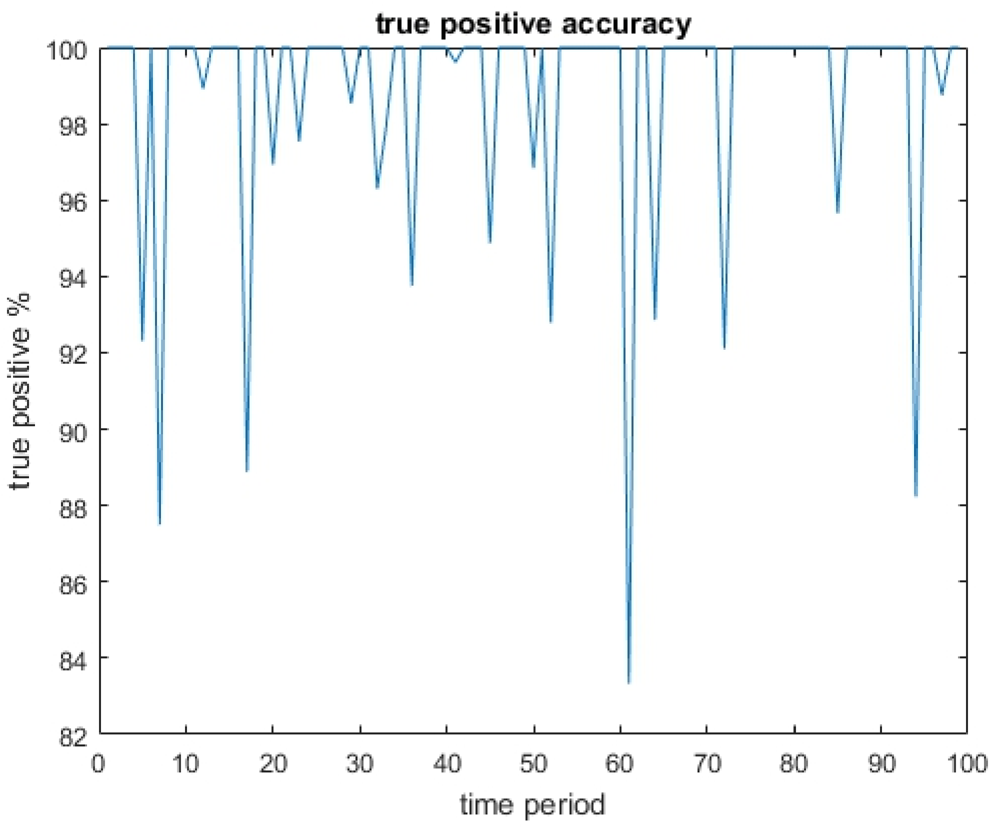


Figure 5: Treu positive Accuracy

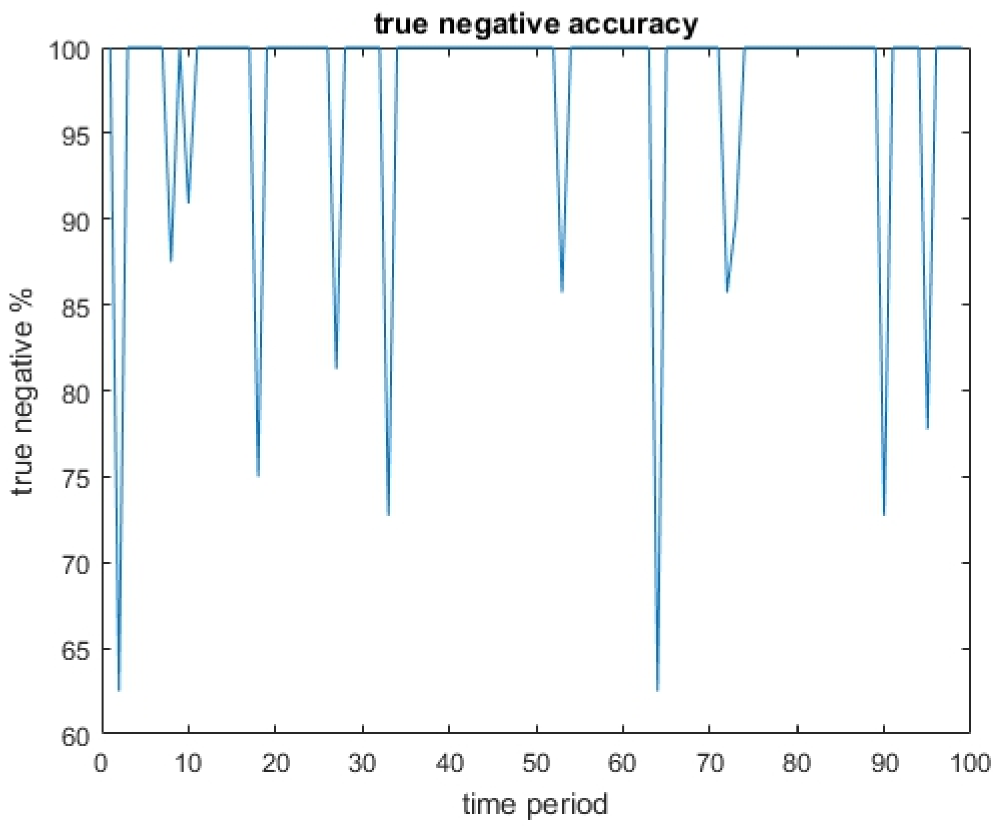


Figure 6: True Negative accuracy

8 Results and Further Research

Our research currently considers link prediction only in the coauthorship domain. In future, we would like to consider a number of datasets from different domains to better understand the link prediction problem. We would also like to define a degree of confidence for link prediction instead of providing a hard binary classification. Moreover, our current attribute set does not have any attributes that capture causal relationships. It is possible that some of the attribute values that we consider are time dependent, i.e. their values should be evaluated by using different weights for different years. In future, we like to consider these kind of attributes. There are online social networks, such as LinkedIn (http://www.linkedin.com) and Friendster (http://www.friendster.com), where they will be very useful. These online networks would like to predict which users would share common interests. These interests are likely to change over time which would affect the likelihood of a link between two users. This is similar to keeping

Citation

I. Bolun C., Fenfen Li, Senbo C., Ronglin Hu, Ling Chen Link prediction based on non-negative matrix factorization 2017

II. Chuanting Zhang, Haixia Zhang, Dongfeng Yuan and Minggao Zhang Deep Learning Based Link Prediction with Social Pattern and External Attribute Knowledge in Bibliographic Networks

III. Jan Miles Co* and Proceso Fernandez Time-Series Link Prediction Using Support Vector Machines 2017

IV. Kyle J., Wayne Lu Application of Machine Learning to Link Prediction, 2016

V. Marin A. , Wellman B. The SAGE Handbook of Social Network Analysis 2011

VI. Mohammad H., Vineet C., Saeed S., and Mohammed Z. Link Prediction using Supervised Learning

VII. Qinxue Meng Paul J. Kennedy Using network evolution theory and singular value decomposition method to improve accuracy of link prediction in social networks 2012

VIII. Theodoros E., Massimiliano P. WORKSHOP ON SUPPORT VECTOR MACHINES: THEORY AND APPLICATIONS