Probabilistic Temporal Process Model for Knowledge Processes: Handling a Stream of Linked Text

Marko Grobelnik, Dunja Mladenić, Jure Ferlež
Department of Knowledge Technologies
Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
E-mail: {marko.grobelnik, dunja.mladenic, jure.ferlez}@ijs.si

ABSTRACT

The paper presents an approach to modelling the data obtained from an observed environment driven by knowledge processes. It is based on the proposed a formalism for presenting probabilistic temporal process model consisting of three major components: (1) background knowledge (in the form of ontologies), (2) observed data (in the form of a stream of data items represented in different data modalities and possibly enriched with background knowledge) and, (3) objectives to optimize (providing guidelines for analytic techniques). The goal is to enable maintaining a data structure - to store, summarize and respond to a wide variety of queries about the observed low level data and about information and knowledge derived from the process. The formalism is realized in software components. Its functioning is illustrated on three scenarios: personal email, corporate email and document collections. The resulting platform is called TNT (Text-Network-Time) according to the main data modalities being addressed within the software.

1 INTRODUCTION

Communication today is greatly supported by electronic media meaning that there is potentially a trace of that communication to be found in electronic form. In we focus on communication inside organizations, the usual practice in organisations is to have more or less well defined formal business processes, informal side of an enterprise life is far from definitions which would allow formal approaches. In a way, we can say that the informal part of the enterprise's life is everything that is not regulated by formal business processes. This means that we can potentially have difficulties even describing what the informal part is, not to mention difficulties when defining, monitoring and analyzing processes within an organization beyond the formal rules.

We can capture the informal part of an enterprise: (a) in a top-down manner where informalities could be described by some kind of rules and logic formalisms, and (b) in a bottom-up data-driven manner, where the goal is to model the processes by observing a life of an enterprise from the side where people leave their traces (in the form of some kind of log files such as, communication records, emails, documents, search logs, etc.). In this paper we focus on the bottom-up/data-driven type of modelling where the goal is to perform automatic discovery of regularities in a functioning of an enterprise based on the available observed data in addition to background knowledge.

Probabilistic temporal process model for knowledge processes is defined here to cover different levels of data complexity where time is an inherent property of the data regardless its complexity. By data complexity we mean here data ranging from simple record of action over records equipped with additional textual description to network of records (where both records and links can have additional textual description). The resulting class of models (modelling temporal processes) is based on an underlying data structure that enables capturing information provided as text, network and time (TNT). In the basic setting the data structure is a timeline of simple records where the main goal is to find patterns leading to identification of tasks that the user is performing and using the identified tasks to predict/suggest to the user the next action (system and/or information source that the user might like to access next). However the timeline of records can be a timeline of documents where the underlying data structure should enable compact storage and representation of text to support search and visualization. Moreover, the records timeline can contain links between the records based on some record properties forming a dynamic graph/network. In that case, mining can be performed on nodes of the graph and/or links of the graph (e.g., taking into consideration neighbourhood of a node).

The main contribution of this paper is the proposed approach based on a unifying formalism being able without significant modifications to deal with different complex scenarios within enterprises. The formalism is designed in a way which allows reusing of appropriate analytic methods from the areas like machine learning, data mining, and social network analysis while preserving enough expressive power of generated structures for modeling the targeted real life situations. The rest of this paper is organized as follows. Section 2 gives a brief description of related work, while Section 3 presents the proposed approach. Implementation of the proposed approach is given in Section 4, while Section 5 and Section 6 give details via describing the data structures and process modelling. The paper concludes with discussion in Section 7.

2 RELATED WORK

Modelling knowledge processes as addressed here assumes that the user is a central actor in the knowledge process accessing some systems/tools and some information sources (data). Probabilistic temporal process model for knowledge processes will be generated from (a) a stream of captured actions captured and (b) relevant background knowledge.

Stream of actions can be seen as a sequence of actions that are potentially connected in more complex actions forming tasks. The area of research dealing in particular with these kinds of scenarios is a subfield of Data Mining called Stream Mining [1, 5], the area got popular as the amount of data increased and the usual scenario where that data was analysed in batch mode was not appropriate anymore. Analogous to Data Mining [4, 8] Action Mining can be used to describe the observed data or for prediction. Probabilistic temporal process model, as defined here, enables both descriptive and predictive mining. In descriptive mining, actions (described by some properties capturing nature of the observed processes) can be described in a more general or coherent way, causal analysis can be performed on them, and anomaly detection can be applied. In predictive mining based on frequent sequences of actions possibly useful patterns can be first proposed to the user to guide identification of tasks. Then each of the identified tasks can be modelled (e.g., by a task template) and applied in the future activity of the user to predict the next user's action. In the above scenarios scalability is an important issue which needs to be addressed separately – the methods dealing with streams are typically able to take data transactions from a stream and change the resulting model incrementally with a small cost per data transaction. Generally we distinguish methods maintaining one global model of the stream data and the ones with many local models (which could act as one big model as well).

Depending on a specific scenario there may be some additional relevant knowledge (commonly referred to as background knowledge) available. Background knowledge can be of rather general nature (such as, an ontology that is describing possible actions) or specific (such as, description of experts skills or some relation between the experts e.g., being frequently in the same project team). The process of action mining may however require generation of background knowledge from some related data. For instance, automatically suggesting experts for specific topics based on authorship of internal enterprise documents on the topics. The main function of background knowledge in this work is to enrich the measured data from the target environment. Namely, the data we are receiving need some extra explanation which is not a part of the measurements – additional information/knowledge in the form of databases, rules, ontologies etc. should therefore serve mainly as data interpretation facility and a way how to bring semantics into the measurements. For example, this can mean extending the existing data records coming from measurements with extra features (based on background knowledge) which will be used for building better statistical models.

3 APPROACH DESCRIPTION

The proposed approach to modeling complex dynamic systems uses the data obtained from an observed environment driven by knowledge processes. The main dimensions along which the approach is being developed are the type of input data the approach is able to deal with, scalability issues and background knowledge for interpretation of the observed data. The main data modalities being used are structured content (e.g., relational bases), graphs extended to networks (e.g., social networks), and textual content (document databases). The data items are coming into the system through time opening the temporal dimension when dealing with the data. For additional interpretation (e.g., data enrichment or introducing semantics) of the data we introduce databases or ontologies which further enrich the observed data. Scalability is achieved by a careful selection of appropriate analytic methods which assure appropriate time and space complexity – the approaches is designed to cover everything from online scenarios (mining streams of data) to batch style processing. Architecture of the proposed approach is shown in Figure 1. The top most object is an environment ("System") including one or more observable and measurable entities ("Entity") whose activities are generalized ("Process Model").

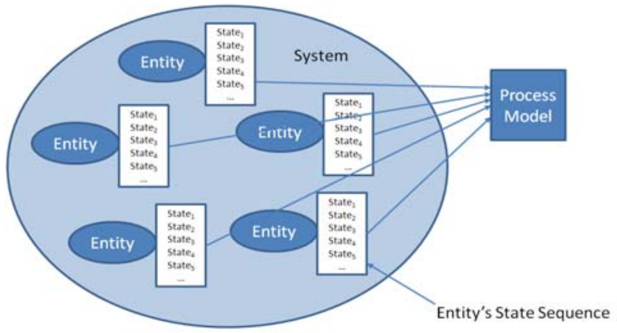

Figure 1 *Architecture of the proposed process modelling.*

The environment "**System"** can be in general anything with some observable internal dynamics. This includes environments like personal computer desktop, e-mail client, document or web server, news forums, web 2.0 portals, social relationships of any kind, agent based systems, corporate business processes, etc. It is not necessary to have a complete understanding and observation of the environment – we should acknowledge that there can always be some hidden parts having influence on the dynamics within the environment.

Within the environment we expect to have one or more (potentially a very large number of) entities (of type "**Entity**") which interact between themselves, with a hidden or visible internal architecture and can have interaction with the world outside the observable environment. Each entity should have a well defined state which captures relevant measurable information about the activity – state of an entity in general cannot capture everything needed for complete understanding of the dynamics, but rather what is practical to measure and analyse. A state in general consists of a set of variables representing different aspects of the measured entity. Each entity has a log of past activities represented as a sequence of past states which serves for further analytic purposes. Examples of entity logs are mouse movements, activated applications, incoming/outgoing emails,web server log files, news forum postings, instance messenger logs, etc. All the entity logs include lots of information which is unmanageable without proper abstraction. The goal of the presented architecture is to construct abstractions ("**Process Model**") in the form of processes describing entity logs in a compressed form understandable for human reading and reusable for various machine applications. The idea is to analyse the data generated by entities and construct a model in the form of patterns, regularities and other kinds of rules for different kinds of tasks like descriptive & exploratory analysis, causal analysis, anomaly detection, prediction etc. The discovered models can then be further used in higher level applications for e.g. optimising personal efficiency on a desktop, for efficient communication over email or instance messenger, for competence discovery within an enterprise, for understanding of informal aspects of an enterprise etc.

4 SYSTEM ARCHITECTURE

The implemented system called TNT (Text-Network-Time), is designed as a set of components maintaining server side functionality. It consists from three major chunks of the functionality: creating TNT database, maintaining the TNT database, querying the TNT database. At the core of the TNT design is a specialized data structure supporting "dynamic networks" enriched with "dynamic content". This structure is able to respond to queries with temporal, network and content constraints. The main functionality of the TNT system is to deal with event processing, events analysis and extraction of abstract event patterns out of the data in a way, which enables accommodating the size and the nature for the streaming data which could be expected within corporate setting. It is expected this to be in the range of ten of events (e.g. emails) and queries per second.

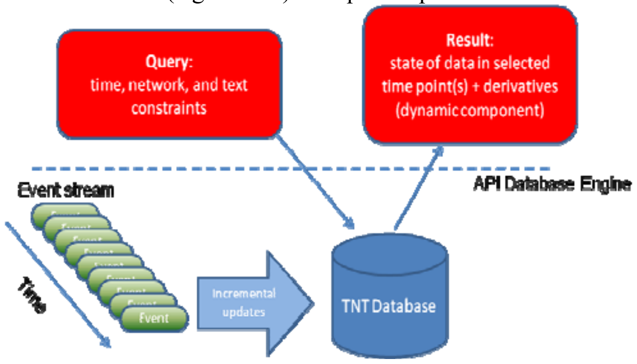
Query:
time, network, and text constraints
Result:
state of data in selected time point(s) + derivatives (dynamic component)
API Database Engine
Event stream
Time
Event
Incremental updates
TNT Database

Figure 2 *Architecture of the TNT system implementing the proposed approach.*

The usual relational databases (such as SQL Server) are not suitable for these kind of tasks because of their transaction processing nature which does not efficiently support the large semi structured data (such as networks and text). Closer to the data structures we are using in TNT database are the inverted index databases for textual information retrieval. However, these have a problem of not supporting temporal queries and networks (graphs), but only support indexing over text. In other words - in the same way as the specialized databases use customized data structures (like textual search, image search, OLAP/data warehouses, triple stores, ontologies, etc.) we use specialized data structures for dealing with the type of operations we will need for mining and searching the kind of data we have in focus (dynamic social networks, dynamic content).

The TNT database will process the events originating in the knowledge workers desktop environment. The TNT database is directly using data structures defined in Section 5. The most important are the dynamic networks enriched with content/textual part with limited queue data structures containing the temporal information. The system is designed as a set of components maintaining server side functionality. It consists from three major chunks of the functionality (see Figure 2): (1) **TNT database** – a data structure allowing other components to insert and query the event data coming into the system. (2) **Maintaining the TNT database** – this includes creation of the TNT database and incremental updates of the TNT data structure for each event coming into the system (Figure 2). It is important to note that the efficient implementation of the incremental updating is the key for the scalability of the system and directly influences efficiency of the throughput of the events. Amortized complexity [**Error! Reference source not found.**] of the updates within our system is constant in time and linear with upper limit (based on the parameters of the system) in space. Amortized complexity means averaging the time and space complexity over the worst case internal infrastructural operations (such as resizing etc). In TNT updates are happening with constant time amortized complexity, while space amortized complexity is linear with the respect to the system parameter which limits the time window of data being monitored. Internally the TNT database maintains the information on a social network between actors. Every actor is described with information on the content, social network and time of the events associated with the actor. Additionally, the content and temporal information on communication between any two actors is maintained in the links of the social network. The TNT is designed to consume one dense stream of events which affect the internal data structures. For instance, TNT is able to consume email events between people inside an organisation in a centralized manner. Each individual event triggers update of the right part of the content, social network, and temporal internal structures of TNT, which describe the actors, the content of communication and time constraints of the email event. We tested the TNT database on a stream of email events generated out of the ENRON dataset [2]. The TNT system running on 2.8GHz machine was able to consume the whole stream of 500 000 email events between 150 people in approximately 15 minutes. (3) **Querying the TNT database** – querying is conceptually similar to other client-server types of architectures, the main difference is in the structure of the query consisting from three major parts: temporal part, network part, and textual part. Therefore, each query isolates network part and content part within the data and shows dynamics of the data within the specified temporal constraints (examples will be shown in the examples section). The resulting output of a query consists from state (network and textual part) of data in different time point (based on the temporal part of the query) and corresponding derivatives showing dynamics (trends) of data.

5 DATA PRELIMINARIES

The key element for modelling knowledge processes is the availability of data within enterprises (where project case studies are just a good sample of situations on the market). In the ideal case, the data would capture all the activities on different levels of an organisation which would allow reconstruction of most processes and other relevant concepts to understand how the organization is functioning. But unfortunately, this is impossible to achieve for many practical reasons. After some review of what kind of data is realistic to retrieve (like availability) and what type of data is manageable (like data complexity, scale, granularity), from technical point of view, we concluded to use some of the fundamental data types on which we have define characteristic operations.

The data can be of different types (modalities). The most typical data types are the following: structured records (e.g. databases), textual data (text documents with some or without structure), and networks (set of connected objects and their relationships). Each of the above data modalities allows characteristic set of operations which could be used for extracting higher level constructs from data. Each of the data types has inherent temporal nature, which means that structured data records and documents are coming in streams and networks have dynamic nature. We have defined the data model for the structures we expect to be dealing with in a way which allows relatively straightforward implementation in programming languages.

Any data can be of atomic or structures types: **DataType = AtomicDataType | StructuredDataType.** Atomic data types are the ones known from programming languages – in the description below we won't go into the details of lexical representations for each of the atomic data types but rather stay on the level of informal description: **AtomicDataType = Void | Boolean | Integer | Float | String.** Each of the atomic data types has defined set of operations which are well known from algebra and programming language practice. Structured data types are composed from the atomic data types and other structured data types. **StructuredDataType = Pair | KeyVal | Tuple | Vector | Stack | Queue | Set | Map.** All of the structured data types require additional subtypes which need to be specified when the data type is getting instantiated.

Complex data types consist of basic data types and implement the key data structures for monitoring processes within enterprises. We tried to restrict the selection to the minimal possible set of data types covering typical data modalities one can find in enterprise environments. We identified three major types of data which we use in a static and stream context (meaning either we have data available as a static dataset or the data is coming in a stream).

Structured records as found in databases represent the most basic way of representing data in an environment. Typically all the standard data of an enterprise would be represented in this way – this corresponds to the data stored in relational databases, indexed in various ways and managed by DBMS type of systems.

Textual data with potentially some additional structure (in a form of e.g. documents, web pages, presentations) represent most of the unstructured data retrievable from enterprise document servers. This type of data represents core of the less formalised part of the enterprise data. By extracting data from text and putting it into other structured forms (like structured records or networks) we make steps towards manageability of the less formal information.

Networks are the most generic formalism to describe structure in the data. The basic underlying structure for a network is a graph with additional information attached to the vertices and edges of a graph. By structuring the information in different ways on a network we can represent very complex situations in an environment. In addition to the standard definition of a static network we will define also a dynamic version of it, which changes through time which will represent basis for monitoring processes within an enterprise.

6 PROCESS MODELING

Process modeling involves some process and modeling methods applied on the process. This Section defines both of them in the light of the proposed approach.

6.1 Process definitions

A "process" is often used as an abstraction for a sequence of events. Processes with some particular meaning would often get a special name denoting the effect the characteristic sequence of events has on the environment. For different contexts where the notion of process has its meaning see <en.wikipedia.org/wiki/Process>. A process always happens to an entity **Entity** which operates within a system **System**. **Entity** is described by its state **State** which is usually described by a set of variables of a type **Tuple** denoting various characteristics of the observer **Entity**.**State = Tuple**, where variables within a Tuple describe state of an Entity. A process is a sequence of states in which the observed entity appears. In the case of discrete time (being reasonable assumption in our case) we can say that the sequence of states is a **Vector** of the type **State. StateSequence<State> = Vector<State**>, where the states are order in time Each state in the sequence is derived by the following formula: **StateSequence[t=0]** = initial state where state variables get initial assignment of values. **StateSequence[t+1] = NextState(Process[t], Action, BKnowledge).** Where an **Action** can be either an empty value or of a type **Tuple** generated by an external source within the **System**. Typically, an **Action** could be an asynchronous event which influences the state of the observed **Entity**. **Action = Tuple.** Background knowledge represented by the parameter **BKnowledge** serves as an additional (mainly static) resource for interpretation of the **Process**, **State** and **Action** within the function **NextState**. **BKnowledge** can appear in the form of a database, set of rules or even as an ontology providing semantics for the data being involved in the description of states within sequences and further on within processes. In this formalization we will avoid specifics of the background knowledge – each example of the **NextState** function in the next sections will provide its own specific type of additional information needed to describe the states and processes. Operationally **BKnowledge** can be understood as a generic query function returning values on the requested (domain specific) queries. Having the data collected in the form of one or several **StateSequence** structures (e.g. one for each **Entity** within the **System**) we can build a model of the data by using analytic techniques such as statistics, machine learning, data mining etc. The model is expressed in a formalism (hypothesis language) supported by the selected analytic method (like rules, decision trees, linear or non-linear functions, etc.). The purpose of the model is to abstract or summarize the analysed data into a shorter description which typically probabilistically resembles the structure and properties of the data. Specific type of modelling and selection of the analytic methods depends on the task (described in the next section). From **StateSequence** data construct **ProcessModel** via modelling using BKnowledge and **Target** value: **ProcessModel**$_{\textbf{Target}}$ ← **Modelling(AnalyticAlgorithm, Vector<StateSequence>, BKnowledge, Target)**. **ProcessModel** is further used in any situations as independent functions getting on the input data from a **StateSequence** and returning the **Target** value which were modelled within the **Modelling** phase:**Target** ← **ProcessModel**$_{\textbf{Target}}$**(StateSequence, BKnowledge).**

6.2 Process Mining

Analysis on the top of the event sequences can be done in various ways. Many areas of science are actively working in subfields of analysis of processes which are mainly determined by the structure and properties of the **State** description of the observed **Entity** and by the language used to describe **NextState** function.

First example of process modelling is from mathematical analysis. Differential equations are well know mechanism for describing processes for the cases where the **State** is described with continuous variables and the function **NextState** is described in analytical language. In the same way, but on the other side of the spectrum within the areas of data mining there is a family of algorithms on "Mining Frequent Episodes" [11] which operate on a set of binary variables and the language for describing **NextState** function is "association rules formalism" (if-then rule).

An important class of processes, which will be of particular interest for our work, are situations where the **State** encodes a network (of a type **Network** or **DynamicNetwork**) and by the change of state (function **NextState**), the observed **Entity** moves from one node to another. Apart from encoding a **Network**, a **State** can include in the rest of its variables also other characteristics of the modelled **Entity**. This is entirely domain dependent and can cover wide range of situations. The importance of this class of processes is relation to process diagrams which are widely used within business process modelling. Diagrams, which can be easy represented by an instance of a type **Network** and corresponding **NextState** function can describe broad class of situations which we can envisage in process mining.

Two typical scenarios for analysing processes are: (1) Analysing the data appearing within the **StateSequence** sequence. This allows finding regularities within the sequences without any specific assumptions on the structure of data etc. This kind of scenario allows mainly understanding of the process event sequences but not other tasks. (2) Identification of the **NextState** function is the most common approach to the analysis of the process data. The goal is to capture specifics of the process within the model by which we describe the function. For certain classes of models (defined typically by the language used for the description of the **NextState** function) we can efficiently calculate the function while for some other classes it can be much more difficult or almost impossible. Usually we are trading between the computational efficiency of the reconstruction of the function versus expressivity of the language for describing **NextState**. Main tasks that we are addressing by process modelling are process description (identify main regularities within the processes and describe them in a formal language), causal modelling (reconstruct causal chains of state sequences includign a subtask of root cause analysis), anomaly detection (identify unusual situations) and prediction (model to predict future unseen situations).

On the top of the structures defined in the previous sections like **Process**, **StateSequence**, **DynamicNetwork**, **Network**, **Document**, etc. one can use a broad class of methods from research fields like machine learning, data/text/web mining, statistics, time series analysis, social network analysis, computational linguistics, etc. Following the structure from the previous section, we can assign some more relevant analytic methods to solve each of the tasks. The assumption here is to have on the input the structure **StateSequence** (possibly enriched with **BKnowledge**) upon which we execute some of the analytic methods to solve the task. Most of the tasks deal with the reconstruction of the function **NextState** or with analysis of **StateSequence** data.

In process description we would like to explain and understand the functioning of the function **NextState** operating in the typical situations described in the **StateSequence** data structure. Main classes of methods for decomposing the **StateSequence** data into some kind of structure are so called unsupervised methods like clustering and other eigenvector decompositions. These kinds of methods provide insights into the structure of the data and together with additional domain knowledge being used for exploratory data analysis enable deeper understanding on what is going on within the data. These methods are often an input for data visualization methods which typically generate graphical summaries of the data. An example of such descriptive analysis would be analysis of a corporate email server an approximate organigram of the analysed organisation based on the sub-communities discovered in the analysis [7].

In causal analysis we typically need to construct inverse of the function **NextState**, where the goal is to predict which state was preceding the current state. First, we need to construct (either from data or domain knowledge) set of discrete states in which the process can appear. Further, with the analysis of the past data of process behaviour we construct models of relationships between the discrete states which enable causal analysis. Typical scenario we try to solve is given an event, causing the process to appear in a certain state, to generate list of other states which most likely preceded our current event to happen. The key here are the methods to detect relationships between the states – this can be done in various ways (depending on the problem we are solving). The most obvious one is correlation between states and events that happen where correlation represent probabilistic relationship between the states. In the presence of additional information we can build local predictive models for "predicting" which path to follow from a state backwards to the cause. Another option is also to build global model of relationships via Markov models where we can test how likely a certain sequence of actions could have happened [10]. Example of causal relationships are probabilistic Markov-network style if-then rules which establish probabilistic relationship of preceding events based on the later sequence of events. **If** Device-123 failed, **then possible causes are**: failure of Device-456 (with probability 0.7), failure of Device-567 (with probability 0.2), failure of Device-789 (with probability 0.1).

In anomaly detection we generally apply statistical measures for measuring surprise or probability of a certain sequence or an event to happen. In a general case, we compare a particular probabilistic distribution with a general one and if it is statistically significantly different then we report an anomaly. With an additional domain knowledge which provides clearer model of behaviour of the observed system we can detect anomalies in a more accurate way. Example of anomalies in the scenario of monitoring computer's desktop activities of the user could be unusual usage of the resources based on the running applications. Unusually high usage of certain resources could be an anomaly caused by a virus or malfunction of the application. Anomalies are often reported as comparison between what is usual and in what way the anomalous situation differs from that. In the application of observing corporate business processes (being typically in the form of process diagrams) an unusual increase of activities in the part of the diagrams where intensity is not as high. This kind of information could be a signal for management to react in a proper way. In the application of observing corporate e-mail server, it could be a change in the intensity of communication between some project group members which can have a cause in problems in their relationships which can further lead to problems on the observed project.

In prediction we predict which state will be the next one in the sequence of states within the **StateSequence** structure. In this task we typically take a window of recent states and try to predict most probable next state. The models we build for prediction are typically built locally for each state in the vocabulary of states. To solve the task we have available a broad range of predictive modelling methods from machine learning and data mining. The methods being used the most in the recent years for predictive tasks are Support Vector Machines (SVM) [3] and Conditional Markov Fields (CRF) [9]. An example of prediction can be within the application on personal e-mails where the goal is to predict when we might expect a response to our e-mail sent to a certain person.

7 DISCUSSION

We have described an approach to modelling complex dynamic systems by modelling the data obtained from an observed environment that is driven by knowledge processes. It is based on the proposed a formalism for presenting probabilistic temporal process. The main functionality of the developed TNT system implementing the proposed approach is to deal with event processing, events analysis and extraction of abstract event patterns out of the data in a way, which enables accommodating the size and the nature for the streaming data which could be expected within corporate setting. The TNT system pipeline currently consist of utilities which enable the system to process the email events' data and metadata, to store it inside the TNT database, and use queries which report on temporal, content and social network aspects of the data and meta data about email stored inside the TNT system.

8 ACKNOWLEDGEMENTS

This work was supported by the Slovenian Research Agency and the IST Programme of the EC PASCAL2 (IST-NoE-216886) and ACTIVE (IST-2008-215040).

References

1. Aggarwal, C., Data Streams: Models and Algorithms. In Advances in Database Systems, 2006.
2. R. Bekkerman, A. McCallum, and G. Huang. Automatic Categorization of Email into Folders: Benchmark Experiments on Enron and SRI Corpora. CIIR Technical Report IR-418 2004
3. Cristianini, N., J Shawe-Taylor, J. (1999) An introduction to Support Vector Machines: and other kernel-based learning, Cambridge University Press
4. Fayyad, U., Piatetski-Shapiro, G., Smith, P., and Uthurusamy R. (eds.) (1996) Advances in Knowledge Discovery and Data Mining. MIT Press, Cambridge, MA, 1996.
5. Gama J., Gaber, M.M., (2007) Learning from Data Streams: Processing Techniques in Sensor Networks
6. Grčar, Miha, Mladenić, Dunja, Grobelnik, Marko. User profiling for interest-focused browsing history. In Proceedings of the 8th international multi-conference on Information Society IS 2005. Ljubljana: Institut "Jožef Stefan", 2005, pp. 182-185.
7. Grobelnik, M., Mladenić, D., Fortuna, B., Ontology Generation from Social Networks. In Semantic Knowledge Management: Integrating Ontology Management, Knowledge Discovery, and Human Language Technologies by Davies, Grobelnik, Mladenic (eds.), Springer, 2009
8. Hand, D.J., Mannila, H., Smyth, P. (2001) Principles of Data Mining (Adaptive Computation and Machine Learning), MIT Press.
9. Lafferty, J.D., McCallum, A., Pereira, F.C.N. (2001) Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data, Proceedings of the Eighteenth International Conference on Machine Learning.
10. Manning, C. Schutze, H. (1999) Foundations of Statistical Natural Language Processing - MIT Press. Cambridge, MA.
11. Mannila, H., Toivonen, H., Verkamo A. (1995) Discovering frequent episodes in sequences. In Proceedings of the First International Conference on KDD.