Navigation Recommendation On Knowledge Artifacts

Jörn David
Institute of Computer Science I1, Technical University Munich
Boltzmannstrasse 3, 85748 Garching, Germany
david@in.tum.de

Abstract: The central problem addressed by this interdisciplinary paper is to provide navigation recommendation on knowledge artifacts represented as nodes of a graph. Actual user navigations on those graphs are captured, which reflect the browsing behavior and interests of many users. The observed navigations are exploited to predict a sequence of artifacts that are relevant to the user. Existing approaches to navigation recommendation are mostly based on symbolic methods like *Association Rule Mining* (ARM), which cannot incorporate the semantics of the entities to be recommended, since their content is not considered at all. Furthermore agile software development principles entail several implications on the desired machine learning support.

We suggest a connectionist approach based on a *Recurrent Neural Network* (RNN) in conjunction with rich content representation to provide navigation recommendation. Our ensemble of machine learning methods is also able to generalize onto unseen artifacts – meaning those that have not been present in the training set. Advanced text analysis methods such as *Latent Semantic Indexing* (LSI) are employed to consider the textual attributes of these artifacts. The proposed technique combines the benefits from both *content-based* and *collaborative filtering* techniques, since the navigation preferences of many users are taken into account. The observed navigation histories are fed into the neural network, which exploits the possibly heterogeneous content of the presented artifacts in order to predict a sequence of related artifacts. We applied our recommendation technique to a publicly available database of web page navigations based on representative log files. The evaluation showed that every second artifact was properly predicted by our machine learning approach using rich content representation.

1 Introduction

As subarea of knowledge management, knowledge sharing is an activity through which information, skills or expertise are exchanged among the members of an organization [NMR+03]. This paper focuses especially on the provision of skills and expertise in form of *control knowledge* [UK95] that becomes manifest in user navigations. Navigation recommendation is a form of user assistance that assists the user with the search for relevant artifacts or the exploration of an artifact space. Normally certain types of users prefer or require certain types of artifacts like documents, manual pages, system models, multimedia items etc., but dislike or ignore other ones. The users exhibit individual behaviors when traversing the navigation space, driven by their different information needs and intended tasks. The central problem of this paper is to learn the users' information needs from the visited artifacts by *learning by example*. Users navigate in an associative way by perform-

ing sequences of navigation operations to solve a knowledge-driven problem, for example trying to fix a bug by browsing through a software documentation. The goal of navigation recommendation is to predict a *sequence of relevant artifacts* based on the observation of the hitherto user navigation on the knowledge graph, which is detailed in the next section.

A lot of research has been done in the field of navigation recommendation that resulted amongst others in *content-based* and *collaborative filtering* [LSY03] techniques. As opposed to our rich content approach, a symbolic realization of collaborative filtering based on *Association Rule Mining* was given by Fu et al., who tried to exploit the tacit knowledge incorporated in the navigation history [BFH00]. Fu proposes an information recommendation system called ***SurfLen***, which suggests interesting web pages to users. Association rule mining as underlying data mining technique is used to discover *frequent 2-itemsets* containing web page URLs like $\{\{p, p_1\}, \{p, p_2\}, \ldots, \{p, p_n\}\}$. When the user is reading page p, then the associated pages $\{p_1, \ldots, p_n\}$ are recommended. So the items stand for URLs here and itemsets are ranked by an intersection measure $rank(U_i, S_j) = |U_i \cap S_j|$ between users' browsing histories $U_i = \{p_1, \ldots, p_m\}$ and mined frequent itemsets $S_j = \{p'_1, \ldots, p'_m\}$. The best k itemsets according to the $rank$-measure are recommended, when the current browsing history is matching one of the stored ones. Such a symbolic approach cannot incorporate the semantics of the entities to be recommended, since the content of the web pages is not considered by the URL-based representation and thus no similarity measure is imposed on these nodes. This deficiency can be solved by considering a *rich representation* of artifacts instead of reducing them to meaningless identifiers.

2 Machine Learning Based on Rich Contents

The software life cycle is a complex and changing process. Modern software engineering methodologies like *Agile Model-Driven Development* that have overcome rigid development processes like the *Waterfall Model* require tool support that takes into account agile paradigms. Machine learning tools must address the basic principles of *agile methods* as far as recommendation functionality is concerned. The following agile principles have direct implications for navigation recommendation on rich knowledge bases.

- **Dealing with Changing and Incomplete Knowledge.** Change is not abnormal but omnipresent and requires continuous retraining of the recommender system, which is more feasible in case of learning from example as opposed to maintain static user profiles. Automatic retraining even leads to a self-adapting recommender system that "follows" the agile development process. To deal with incomplete and uncertain knowledge, the machine learning algorithm must be robust, which is an inherent property of connectionist machine learning methods.

- **Content is More Important Than Representation.** [Amb06] This claim is taken into account by the *multi-representation* of domain objects defined beneath. This technique allows to exploit various types of attributes such as text, categorical, metric, etc. and serves for independence from the respective content representation.

- **Everyone Can Learn From Everyone Else.** [Amb06] This paradigm is especially supported by navigation recommendation, since a project participant can learn from other ones by walking on their paths through the knowledge base, while being proactively guided by recommendations telling how to proceed. Normally a beginner learns from a domain expert, but also an expert can behave like a beginner when faced with unknown sub domains or new processes.

Since in the general case no behavioral rules are available that reveal the browsing behavior of users on a certain knowledge base in form of explicit descriptions, learning from examples is the only way to incorporate knowledge about the users and the application domain. Interests and expertise of the users often become manifest in their *navigation behavior* on the respective knowledge base. The users implicitly reveal their interests and furthermore produce new associations by actually using the system. An example of a tree-shaped knowledge base is depicted by figure 1. Our recommendation technique implicitly

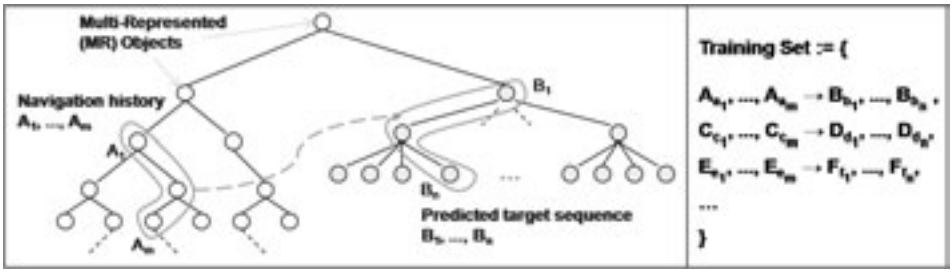

Figure 1: Exemplary user navigation consisting of history and predicted target sequence on a tree-shaped knowledge base. A set of those navigations makes up a *training set*.

addresses the aspect of *collaboration* and knowledge sharing by learning from the navigation behavior of many users, which may be distributed across several teams working on the same project. In a multi-user environment, the navigation histories can be contextualized by the respective user, such that the same sequence of visited objects leads to different recommendations for different users. This can be achieved by encoding the user in the multi-representation of each visited artifact.

Multi-represented objects as constituents of the navigation sequences depicted by figure 1 capture several different aspects of the same domain object. These aspects are called *features* in the field of knowledge discovery, which is equivalent to the denominations *attribute* or *dimension*. A recent example is the encapsulation of all biometric features of a person like voice pattern, image and finger print in one multi-represented object.

Definition 2.1 ***(Multi-Represented Object)*** *A multi-represented (MR) object is an element of a multi-dimensional feature space: $o = (r_1, \ldots, r_n) \in F_1 \times \ldots \times F_n$, where F_i is a feature that can be weighted by a factor $w_i \in [0, 1]$ additionally. Missing values r_i should principally be allowed in the object representation, since modern machine learning algorithms are able to deal with incomplete data. The allowed feature types are:* ***Unstructured text****,* ***Metric*** *(distance measure),* ***Ordinal*** *(ordered) and* ***Categorical*** *(unordered).*

Table 1: Multi-representation of objects by different features. One table line defines the schema of one object, which can have a different number of features, depending on the object type. The available feature types are: *Text*, *Metric*, *Boolean* (a subtype of *Categorical*), *Ordinal* and *Categorical*. *KPI* means *Key Performance Indicator*.

MR-Object Type	**Feat$_1$[Type]**	**Feat$_2$[Type]**	**Feat$_3$[Type]**	...
Requirement	Name[Txt]	Description[Txt]	...	
Issue	Name[Txt]	Description[Txt]	IsResolved[Bool]	Prio[Ordinal]
Picture	Name[Txt]	Caption[Txt]	Histogram[Metric[]]	
Web page	URL[Txt]	Head[Txt]	Body[Txt]	

Each feature type like *categorical* or *metric* requires its own treatment with respect to its possible value range and the appropriate scaling function to be applied during data preprocessing. A concrete instance of a multi-represented object according to the schema of table 1 above would be an *issue ("Issue_abc01", "Solving the bug in the ODBC database connection", false, 8)*. The configuration of the used feature set can even change within the same discourse domain, for example to represent text documents on the one hand and more formal use case elements on the other hand. Each MR-object that contains attributes standing for unstructured text has to be transformed into a machine-processable representation. The textual content of software artifacts is considered by a text mining approach, which provides a rich semantical representation of artifacts and imposes a similarity measure upon them. We apply an advanced text mining method called *Latent Semantic Indexing (LSI)* for computing a low-redundancy numerical representation (vector space model). Thereby a matrix $M_{i,j}$ of keyword frequencies per text unit (e.g. sentences, paragraphs, sections, documents) is spanned. The rows denote the frequency of occurrence for term i in text unit j, which represents the textual content of an artifact. *Stemming* and *stop-word removal* are realized by the *Apache Lucene* indexing and search framework (http://lucene.apache.org).

Definition 2.2 ***(Sequence Prediction)*** *Sequence prediction is a mapping $P : V^* \rightarrow V^*$ from history sequences to target sequences of nodes, where V is the set of all nodes. The nodes of the graph-based knowledge based depicted by figure 1 stand for arbitrary multi-represented objects. The represented object features F_i are also allowed to be continuous $F_i \in \mathbb{R},\ i = 1, \ldots, d$, for example when predicting the remaining time to complete a new software feature indicated by burn down charts.*

Application to Software Engineering Building *Predictive Software Models (PSMs)* by methods from artificial intelligence such as *classification and regression trees (CART)*, neural networks or case-based reasoning is already a big trend in software engineering. There are a lot of research contributions that propose the prediction of the *project success* [Smi07], the *project costs* [She05], the *project duration* and so on.

The machine learning system presented in this article aims at providing recommendations on a document level rather than on source code, especially not on the fine-grained level of operations. For this purpose, the work of Zimmermann et al. [ZWDZ04] might be more appropriate, who applied non-rich but symbolic *association rule mining* to analyze code changes that frequently occur together and thus is able to give recommendations upon

source code fragments. A similar tool that recommends example code from third-party libraries and APIs is *Strathcona* [HWCM05], which operates on a relational database.

Our approach is beneficial when used on rich entities such as issues, requirement documents or communication artifacts consisting of unstructured text or which are even multi-represented by several attributes. As an example, during *controlling* of a software project the responsible project participant might want to know which artifacts have typically been changed together. When a requirement is changed during analysis (Requirements Analysis Document (RAD)) or an issue is attached, the affected use case documents and diagrams also have to be adapted. Figure 2 shows an exemplary landscape of requirements, issues and use case artifacts that are often changed together. The controller can quickly browse

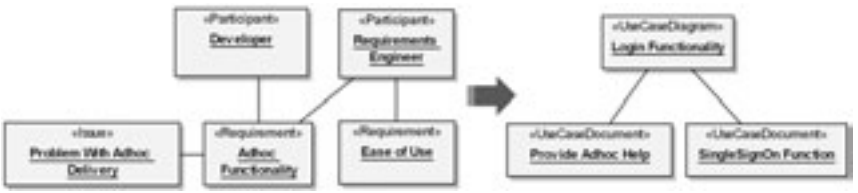

Figure 2: UML object diagram showing different types of development artifacts concerning a guided login functionality, which are frequently changed together. These analysis objects can be interpreted as nodes of a project graph in terms of figure 1. In this scenario, the left partition of objects is sighted by the project controller and the right partition is recommended despite the absence of a hard-wired link between them (missing inter-model link). The project participants are modeled as objects, since the system also exploits their profiles for recommending related artifacts.

through the changed requirements, while the system proactively recommends the relevant use case documents upon content-based similarity *and* the navigations performed by the participants from the analysis team.

3 The Recurrent Neural Network

The *Modular Recurrent Neural Network* (MRNN) is employed as enabling technology in this paper, thus its detailed design is out of scope here. The main purpose of the MRNN is to learn navigation sequences on a knowledge graph and to predict the most appropriate *target sequence* according to the domain-specific user interests. Main benefits of neural networks are their inherent context-representation as well as high generalization capability, flexibility and robustness. The neural prediction is computed on the basis of the recent *navigation history* $\vec{x}_{t-k}, \ldots, \vec{x}_t$ according to the observed user behavior. Figure 3 illustrates the topology and the information flow of the employed recurrent neural network. $\vec{y}_{t+1}, \ldots, \vec{y}_{t+m}$ represents the associated sequence of target nodes, which are predicted to correlate strongest with the observed navigation history. An important characteristics of the recurrent network design is its inherent temporal memory provided by the *hidden state layer* $\vec{s}_{t-k}, \ldots, \vec{s}_{t+m-1}$. Due to the *temporally unfolded network topology*, the MRNN can learn the *sequential structure* of a set of node sequences. Thus the navigation behavior

of many users is explicitly modeled in the hidden state layer. One input vector $\vec{x}_t$ or output vector $\vec{y}_{t+1}$ corresponds to one node of the knowledge graph.

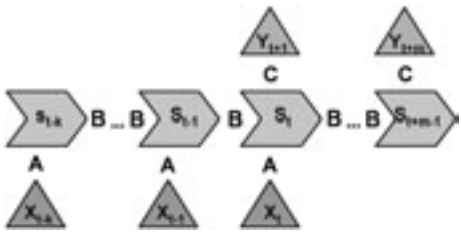

Figure 3: *Schematic topology of the proposed modular recurrent network MRNN. The block arrows indicate the internal state transition $\vec{s}_t \rightarrow \vec{s}_{t+1}$. A, B and C are weight matrices. $\vec{x}_t$ is the external input vector at time t, $\vec{y}_{t+1}$ is the correspondingly predicted output vector. The depicted block arrow direction shows the forward propagation phase.*

In order to process variably long node sequences $\vec{x}_{t-k}, \ldots, \vec{x}_t$, $\vec{y}_{t+1}, \ldots, \vec{y}_{t+m}$ as part of the same training set, we developed a modular neural network. Its technical design is defined by the following recurrent model describing the forward propagation phase.

- A is a ${}^h\mathbb{R}^{d_1}$, B is a ${}^h\mathbb{R}^h$ and C is a ${}^{d_2}\mathbb{R}^h$ matrix. d_1 is the dimensionality of the input- and d_2 is the dimensionality of the output feature space.
- $h = dim(\vec{s}_i)$, (i = t-$k, \ldots, t$+m) is the dimensionality of the state layer. h is independent from d_1 and d_2 and was set to $h = 20$ (experimentally determined) to provide sufficient network resources.

$$\vec{s}_t \quad = \quad f(B\vec{s}_{t-1} + A\vec{x}_t), \qquad \vec{o}_{t+1} = f(C\vec{s}_t) \tag{1}$$

$$\vec{o}_{t+i} \quad \overset{training}{\longrightarrow} \quad \vec{y}_{t+i}, \quad i = 1, \ldots, m \tag{2}$$

As introduced above, $\vec{s}_t \in S$ stands for the internal state at the discrete time step t. The crucial recurrent equation 1 combines an external input $\vec{x}_t$ with the previous state $\vec{s}_{t-1}$ to the subsequent state $\vec{s}_t$, which indirectly depends on all foregoing external inputs $\vec{x}_{t-k}, \ldots, \vec{x}_{t-1}$ *and* internal states $\vec{s}_{t-k}, \ldots, \vec{s}_{t-1}$. In case of supervised network training, the target symbols $\vec{y}_{t+1}, \ldots, \vec{y}_{t+m}$ are known. Here, the activation function is chosen as sigmoid function $f(x) = \frac{1}{1+\exp(-x)}$. During one training epoch, the network adapts at runtime to the individual structure of the respective node sequence with its history and target part. In the operational application of the trained network, two functions are provided:

1. *Recognition and Recovery*: When presenting a sequence of artifacts to the MRNN that equals the *history part* of a learned navigation sequence, the subsequently visited artifacts can be *recovered* and *recommended*.
2. *Generalization*: A virtually unknown sequence of software artifacts is fed into the MRNN, which is then able to predict the most likely related artifacts.

The MRNN operates like a *dynamic system* by explicitly modeling a state layer, which also enables the systematic processing of structured knowledge, which is often used in the knowledge and software engineering domain.

There are still a few *limitations* of the connectionist approach: 1. The length of the navigation history has to be chosen as parameter in order to know when to compute a prediction. Alternatively several recommendations can be computed for several histories of different lengths. 2. Relational training patterns cannot be processed directly, these have to be transformed to functional mappings of history and target sequences.

4 Application to Real-World Navigation Data

We applied our recommendation technique to a knowledge base of web page visitations, since no actual software engineering datasets have been available that reflect the navigation behavior on project artifacts. Microsoft has logged web access data concerning its portal *www.microsoft.com* by sampling and processing the server log files, resulting in 32,700 user navigations on 329 different web pages [BHK98]. The data reflects the usage of the internet portal by 38,000 anonymous, randomly-selected users. For each user, the set of all visited web pages in a timeframe of one week is stored, which is considered as the user's navigation. The web page access data was split up into an arbitrary training set of 2,000 cases and a test set of 1,000 cases for evaluating the connectionist generalization capability. The MRNN is trained on navigation sequences such as $(1081, 1082, 1040 \mapsto 1001)$ or $(1081 \mapsto 1040, 1001)$, where the integers identify a single web page and their textual titles are represented using *latent semantic indexing*. Navigation sequences are not bound to specific users but are representative for all of them. After training, all node sequences where *recovered successfully* when navigating according to the learned history nodes, such that the corresponding sequence of target nodes was recommended properly.

A successful recommendation is given, if a web page visit out of the exemplary subset $H := \{1081, 1082, 1040\} \subset TC$ of a single test case, e.g. $TC := \{1081, 1082, 1040, 1001, 1018, 1083\}$, is presented and the network predicts one node of the complementary set $T := \{1001, 1018, 1083\}$. In general, one node $u \in H$ is presented and one node $v \in T = TC \setminus H$ of the target subset is expected as correct recommendation, which counts as hit and is summed over all test cases. The system was able to deal with unseen web pages, that means such that had never or only partially been presented to the neural network in the training phase. We trained the network on 2,000 navigation examples, whereupon the target nodes of 1,000 test sets could be inferred with a precision of 50.00%.

5 Conclusion

We have introduced a semantic navigation recommendation based on our *Modular Recurrent Neural Network (MRNN)* in conjunction with advanced text mining capabilities. The MRNN enables learning of variably long node sequences on graph-based and domain-

specific knowledge bases. We modeled the users' navigation behavior as paths through a knowledge base of artifacts and trained the network on the observed navigation sequences in a supervised manner. Altogether we have designed and fully implemented a recommender system that fulfills two main tasks: ***1. Learning and recovering of node sequences on graph-based knowledge bases*** and ***2. Prediction of node sequences according to the users' interests and in consideration of their multiple contents***. The learned user preferences are generalized by the MRNN, which predicts relevant artifacts upon *unseen* navigation histories. Connectionist recommendation combined with latent semantic indexing achieved an accuracy of 50% on a Microsoft test set of 1,000 navigation patterns each consisting of history and target sequence. Every second node that was recommended in the evaluation matched the users' interests. Compared with the probabilistic *n-gram recommendation* based on NASA web server logs by Sun et al. [SCWM02], which achieved a precision of about 37%, our technique shows a significantly higher performance.

References

[Amb06] Scott W. Ambler. *The Object Primer: Agile Model-driven Development with UML 2.0*. Cambridge, 2006.

[BFH00] Jay Budzik, Xiaobin Fu, and Kristian J. Hammond. Mining Navigation History for Recommendation. In *Proceedings of the 5th international conference on Intelligent user interfaces*, pages 106–112. ACM Press, New York, NY, USA, 2000.

[BHK98] John S. Breese, David Heckerman, and Carl Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, 1998.

[HWCM05] Reid Holmes, Robert J. Walker, and Gail C. Murphy. Strathcona example recommendation tool. In *ACM Software Engineering Notes*, volume 30, pages 237 – 240, 2005.

[LSY03] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. In *IEEE Internet Computing*, volume 4, 1, 2003.

[NMR+03] Donald M. Norris, Jon Mason, Robby Robson, Paul Lefrere, and Geoff Collier. A Revolution in Knowledge Sharing. 2003.

[SCWM02] Xiaoming Sun, Zheng Chen, Liu Wenyin, and Wei-Ying Ma. Intention Modeling for Web Navigation. In *Proceedings of the 11th World Wide Web Conference*, 2002.

[She05] M. Shepperd. Evaluating software project prediction systems. In *Software Metrics, 11th IEEE International Symposium*, pages 2 pp. –, 2005.

[Smi07] Darja Smite. Project Outcome Predictions: Risk Barometer Based on Historical Data. In *International Conference on Global Software Engineering (ICGSE07), Munich, Germany*. Riga Information Technology Institute, Latvia, 2007.

[UK95] Alfred Ultsch and Dieter Korus. Integration of Neural Networks with Knowledge-Based Systems. In *Proc. IEEE Int. Conf. Neural Networks, Perth/Australia*, 1995.

[ZWDZ04] Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, and Andreas Zeller. Mining Version Histories to Guide Software Changes. In *International Conference on Software Engineering (ICSE 2004)*, 2004.