Sequence Mining for Customer Behaviour Predictions in Telecommunications

Frank Eichinger[1], Detlef D. Nauck[2], and Frank Klawonn[3]

[1] Universität Karlsruhe (TH), Institute for Program Structures and Data Organisation (IPD), Karlsruhe, Germany, eichinger@ipd.uka.de
[2] BT Group plc, Intelligent Systems Research Centre, Ipswich, UK, detlef.nauck@bt.com
[3] University of Applied Sciences Braunschweig/Wolfenbüttel, Department of Computer Science, Wolfenbüttel, Germany, klawonn@fh-wolfenbuettel.de

Abstract Predicting the behaviour of customers is challenging, but important for service oriented businesses. Data mining techniques are used to make such predictions, typically using only recent static data. In this paper, a sequence mining approach is proposed, which allows taking historic data and temporal developments into account as well. In order to form a combined classifier, sequence mining is combined with decision tree analysis. In the area of sequence mining, a tree data structure is extended with hashing techniques and a variation of a classic algorithm is presented. The combined classifier is applied to real customer data and produces promising results.

1 Introduction

Predicting churn, i.e. if a customer is about to leave for a competitor, is an important application of analysing customer behaviour. It is typically much more expensive to acquire new customers then to retain existing ones. In the telecommunication industry, for example, this factor is in the range of about five to eight [1]. Correctly predicting that a customer is going to churn and then successfully convincing him to stay can substantially increase the revenue of a company, even if a churn prediction model produces a certain number of false positives.

Beside the prediction of churn, other customer-related events like faults, purchases or complaints can be predicted in order to be able to resolve some problems before the actual event occurs. The prediction of sales events can be used for cross-selling, where a certain product is offered just to customers who have an increased likelihood to buy it.

In the telecommunications industry the available customer data is typically timestamped transactional data and some static data (e.g. address, demographics and contract details). Transactional data are sequences of timestamped events which can easily be stored in relational database tables. Events can be any kind of service usage or interaction, particularly calls to the company's call centre, for example, complaints or orders.

In the area of data mining, many approaches have been investigated and implemented for predictions about customer behaviour, including neural networks, decision trees and naïve Bayes classifiers (e.g., [1, 2, 3]). All these classifiers work with static data. Temporal information, like the number of complaints in the last year, can only be integrated by using aggregation. Temporal developments, like a decreasing monthly billing amount, are lost after aggregation. In this paper, sequence mining as a data mining approach that is sensitive to temporal developments is investigated for the prediction of customer events.

In Chapter 2 we present sequence mining and its adoptions for customer data in telecommunications along with an extended tree data structure. In Chapter 3, a combined classification framework is proposed. Chapter 4 describes some results with real customer data and Chapter 5 concludes this paper and points out the lessons learned.

2 Sequence Mining

Sequence mining was originally introduced for market basket analysis [4] where temporal relations between retail transactions are mined. Therefore, most sequence mining algorithms like AprioriAll [4], GSP [5] and SPADE [6] were designed for mining frequent sequences of itemsets. In market basket analysis, an itemset is the set of different products bought within one transaction. In telecommunications, customer events do not occur together with other events. Therefore, one has to deal with mining frequent event sequences, which is a specialisation of itemset sequences. Following the Apriori principle [7], frequent sequences are generated iteratively. A sequence of two events is generated from frequent sequences consisting of one event and so on. After generating a new candidate sequence, its support is checked in a database of customer histories. The support is defined as the ratio of customers in a database who contain the candidate sequence in their history.

2.1 Sequence Mining for Customer Behaviour Predictions

A crucial question in sequence mining is the definition of the relationship "S is contained in T" (denoted as $S \prec T$), which is decisive for determining the support of a sequence. Originally, a sequence S is contained in a sequence T, if all elements of S occur in T in the same order [4]. It does not matter if S and T are equal or if one or more additional events are contained in T as well. A strict definition would not allow any extra events in between the events of sequence T, but at its beginning and end. For example, $\langle C \leftarrow B \leftarrow A\rangle \prec \langle X \leftarrow X \leftarrow X \leftarrow C \leftarrow Y \leftarrow B \leftarrow Y \leftarrow A \leftarrow Z\rangle$ is true in the original definition, but not in the strict one as there are two events Y which are not allowed.

In this work, we want to use sequence mining for classification. If a certain sequence of events was identified leading to a certain event with a high confidence, we want to use this sequence for classifying customers displaying the same sequence. If we chose the strict definition of "is contained in", we would not classify

customers correctly who contain a very significant sequence but with an extra event in between. This extra event could be a simple call centre enquiry which is not related to the other events in the sequence. The original definition would allow many extra events occurring after a matched sequence. In the application to customer behaviour prediction, a high number of more recent events after a significant sequence might lower its impact. Therefore, we introduce two new sequence mining parameters: *maxGap*, the maximum number of allowed extra events in between a sequence and *maxSkip*, the maximum number of events at the end of a sequence before the occurrence of the event to be predicted. With these two parameters, it is possible to determine the support of a candidate sequence very flexibly and appropriately for customer behaviour predictions. For instance, the presented example is true if $maxGap = 2$ and $maxSkip = 3$. It is not true any more, if one of the parameters is decreased.

2.2 The Sequence Tree Data Structure

Multiple database scans, which are necessary after every generation of candidate sequences, are considered to be one of the main bottlenecks of Apriori-based algorithms [8, 9]. Such expensive scans can be avoided by storing the database of customer histories efficiently in main memory. In association rule mining, tree structures are used frequently to store mining databases (e.g., [8]). In the area of sequence mining, trees are not as attractive as lattice and bitmap data structures (e.g., [6, 9]). This is due to smaller compressing effects in the presence of itemsets. In our case, as well as in the application of sequence mining to web log analysis (e.g., [10]) where frequent sequences of single events are mined, tree structures seem to be an efficient data structure. In this paper, such a tree structure or more precisely trie memory[4] [11] as known from string matching [12], is employed to store sequences compressed in main memory. We call our data structure *SequenceTree*.

In the *SequenceTree*, every element of a sequence is represented in an inner- or leaf node. The root node and all inner nodes contain maps of all direct successor nodes. Each child represents one possible extension of the prefix sequence defined by the parent node. The root node is not representing such an element, it just contains a map of all successors, which are the first elements from all sequences. Every node, except the root node, has an integer counter attached which indicates how many sequences are ending there.

An example for a *SequenceTree* containing five sequences is given in Figure 1. To retrieve the sequences from the tree, one can start at every node with a counter greater than zero and follow the branch in the tree towards the root node. Note that if the sequence $\langle A \leftarrow B \leftarrow C \rangle$ is stored already, just a counter needs to be increased if the same sequence is added again. If one wants to add $\langle A \leftarrow B \leftarrow C \leftarrow D \rangle$, the last node with the C becomes an inner node and a new leaf node containing the event D with a count of one is added.

[4] Tries are also called prefix trees or keyword trees.

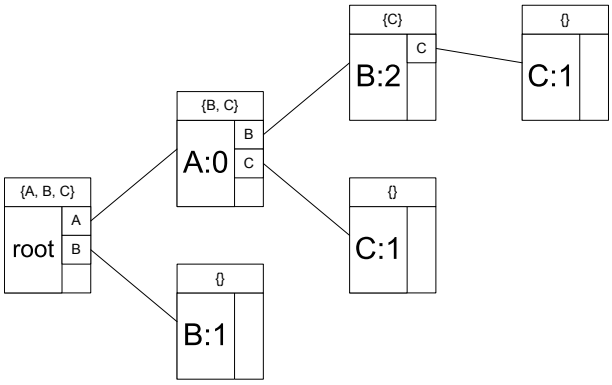

Figure 1. A *SequenceTree* containing the sequences $\langle A \leftarrow B \leftarrow C \rangle$, $\langle A \leftarrow B \rangle$ (twice), $\langle A \leftarrow C \rangle$ and $\langle B \rangle$. The number after ":" indicates the count how many sequences are ending in the node. The hash tables with all subsequent events are denoted by {}.

The compact storage of sequences achieved using the *SequenceTree* is due to two compressing effects:

1. The usage of counters as in [8, 9, 10] avoids the multiple storage of the same sequences. Obviously, the compression ratio depends very much on the kind and amount of data. Experiments with real customer data showed that the usage of counters reduces the memory necessary to store the sequences by a factor of four to ten.
2. Sequences with the same prefix sequence are stored in the same branch of a tree as done in the finite state machines known from string pattern matching [12]. Especially if sequences are long, this technique can reduce the memory needed significantly.

In sequence mining algorithms like in [4, 5] or in the one described in the following subsection, it happens very frequently that a candidate sequence is being searched in a database in order to determine its support. These searches can be very time consuming, even if the database is stored in an efficient data structure. In order to speed up searches in the *SequenceTree*, hash tables are used in every node which contain all events occurring in all succeeding nodes. If a candidate sequence is searched in the tree, the search can be pruned at an early stage if not all events in the searched sequence are included in the hash table of the current node. For example, we want to count the support of the sequence $\langle A \leftarrow B \leftarrow D \rangle$ in the *SequenceTree* from Figure 1. The search algorithm would check the hash table of the root node first. As D is not contained in this table, the search could be stopped immediately. As hash tables provide constant time performance for inserting and locating [13], the maintenance of hashtables as well as lookups do not require much extra time. Also the memory overhead is marginal as it is sufficient to store small pointers to events in the hash tables. In our experiments we measured a speed up of three by utilising hash tables in a *SequenceTree* during a real churn prediction scenario.

2.3 Sequence Mining Using the Sequence Tree

In Figure 2 a sequence mining algorithm taking advantage of the *SequenceTree* is described. This algorithm is based on AprioriAll [4], adopts its candidate generation, but avoids multiple database scans as the database is being loaded into a *SequenceTree* first. In every iteration, candidate sequences $candidates_k$ are generated. Afterwards, the support of every candidate *cand* is calculated in the *SequenceTree* $\mathcal{C}$. Only Sequences exceeding a user defined minimum support *minSup* are kept and returned at the end.

Require: $\mathcal{C}$ (database of customers), *minSup*(sequence mining parameter)
  $L_1 = \{\langle E \rangle \mid support(\langle E \rangle) \geq minSup\}$
  for $(k = 2;\ L_{k-1} \neq \emptyset;\ k{+}{+})$ **do**
    $candidates_k = generate\_candidates(L_{k-1})$
    for all $(cand \in candidates_k)$ **do**
      $cand.count = \mathcal{C}.determineSupport(cand)$
    end for
    $L_k = \{cand \mid support(cand) \geq minSup \wedge cand \in candidates_k\}$
  end for
  return $\bigcup_k \{S \mid S \in L_k\}$

Figure 2. Sequence mining algorithm with $\mathcal{C}$ stored in a *SequenceTree*.

The method *determineSupport*() is responsible for calculating the support of a sequence. This is done by performing a greedy depth first search of the candidate sequence in the $SequenceTree$[5]. Due to the introduced parameters *maxGap* and *maxSkip* which allow a flexible definition of support, the search is not as easy as searches in string matching [12]. The parameter *maxGap* is implemented by skipping up to *maxGap* nodes during the search and backtracking afterwards. Backtracking and searching in all possible branches is necessary, as a candidate sequence can occur in several branches if gaps are allowed. The parameter *maxSkip* requires to perform up to *maxSkip* searches in parallel. Up to *maxSkip* events can be skipped at the beginning of a search. Therefore, a new parallel search is started at every node which is reached by the search algorithm by traversing deeper into the tree.

2.4 Experimental Sequence Mining Results

Sequence mining as described in the previous subsection was applied to real customer data in a churn prediction scenario. The dataset used was artificially sampled in order to obtain an overall churn rate of exactly 4%. A number of sequences were found and for every sequence a confidence value was calculated. The confidence value is a likelihood for the occurrence - in this case - of a churn

[5] All algorithms traversing the tree were implemented iteratively as our experiments showed a significant performance gain compared to recursive implementations.

event. The result was a set of sequential association rules like the following one: "$\langle ENQUIRY \leftarrow ENQUIRY \leftarrow REPAIR \rangle$, confidence = 4.4%, support = 1.2%", meaning that 1,2% of all customers display the pattern with a specific repair first, than an enquiry followed by another enquiry in their event history. 4.4% of all customers with this pattern are having a churn event afterwards. Therefore, we know that customers displaying this pattern have a slightly higher churn probability than the average of all customers. On the other hand, a support of 1,2% means that just a small fraction of all customers is affected. This rule is just one rule in a set of around hundred rules (depending on the predefined minimum support). Only some rules exist with a higher confidence of around 10%, but they affect even smaller fractions of customers. Even if such a rule with a high confidence of e.g. 10% is used to identify churners, this rule would still classify 90% of the customers incorrectly. Therefore, even a large set of sequential association rules was not suitable for churn prediction.

3 A Framework for Customer Behaviour Prediction

Given that more information than just the sequential order of events was available in our application scenario, we built a classifier which is based on sequence mining, but analyses additional attributes with decision trees. These additional attributes are such associated with the customer (e.g., the contract duration), the sequence (e.g., the number of days between two events) and the events itself (e.g., the time to resolve a repair). A similar combination of sequence mining and other classifiers has been successfully implemented in bio-informatics [14]. In the following, we describe a prediction framework (Figure 3) consisting of a model building process and a classification process.

In the model building process, sequence mining as described in the previous section is applied first. Afterwards, a decision tree is induced and pruned for each detected sequence incorporating a number of further attributes. The sequences are saved together with the corresponding decision trees building a combined classification model.

In the classification process, single customers are classified using the classification model. At first, sequences that are supported by the customer's event history are selected from the model. Subsequently, the customer is classified by the decision trees associated with these sequences. The final classification of the customer is computed by averaging all results and applying a threshold value.

4 Experimental Results

The combined classifier was applied to real customer data from a major European telecommunication provider. In this paper, just some results from a churn prediction scenario are presented, even if the model was tested in a number of different scenarios for different events. For reasons of data protection, non-representative random test samples with a predefined churn rate had to be generated. In a three months churn prediction scenario, the combined classifier was first trained

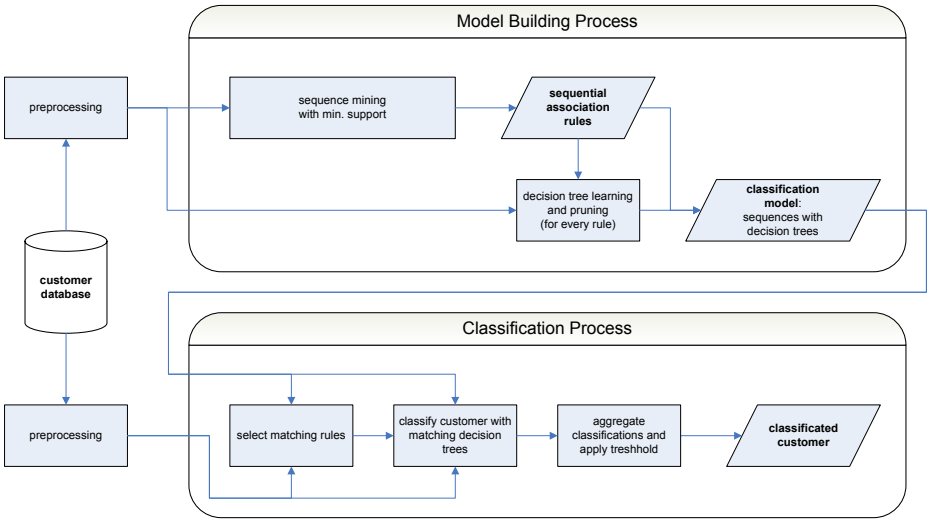

Figure 3. The framework for customer behaviour prediction.

with historic data including all events within one year and then applied to a test set from a more recent time window. The classifier found 19.4% of all churners with a false positive rate of only 2.6%. The gain[6] of this test result – the ratio how much the classifier is better than a random classifier – is 5.5.

It is hard to compare our test results. On the one hand, all results related to customer data are usually confident and therefore they are not published. On the other hand, published results are hardly comparable due to differences in data and test scenarios. Furthermore, most published results were achieved by applying the predictive model to a test set from the same time window (e.g., [3]) instead of making future predictions.

5 Conclusion and Lessons Learned

In this paper we extended a tree data structure and approach for sequence mining. This approach was combined with decision trees in order to form a combined classifier which is able to predict any desired customer event.

In the area of sequence mining, we showed that traditional definitions of support and especially of the "is contained in" relationship are not feasible for customer behaviour predictions. We introduced two new parameters to flexibly specify this relation.

As multiple events at the same time are unusual in telecommunication customer data, we introduced an extended tree data structure and algorithm for

[6] The gain measure is defined as the predictor's churn rate (the ratio of all correctly predicted churners to all customers predicted as churners) divided by the a priori churn rate (the rate of churners in the test set).

mining sequences of single events. We showed that our tree structure in combination with hashing techniques is very efficient.

Our investigations showed that sequence mining alone is not suitable for making valuable predictions about the behaviour of customers based on typically rare events like churn. However, it is capable of discovering potentially interesting relationships concerning the occurrence of events.

Furthermore, our study showed that it is more promising to analyse temporal developments by employing sequence mining in combination with other classifiers than to use only static classification approaches.

References

[1] Yan, L., Miller, D.J., Mozer, M.C., Wolniewicz, R.: Improving Prediction of Customer Behavior in Nonstationary Environments. In: Proc. International Joint Conference on Neural Networks (IJCNN). (2001)

[2] Buckinx, W., Baesens, B., den Poel, D., van Kenhove, P., Vanthienen, J.: Using Machine Learning Techniques to Predict Defection of Top Clients. In: Proc. 3rd International Conference on Data Mining Methods and Databases. (2002) 509–517

[3] Neslin, S.A., Gupta, S., Kamakura, W., Lu, J., Mason, C.H.: Defection Detection: Measuring and Understanding the Predictive Accuracy of Customer Churn Models. Journal of Marketing Research **43**(2) (2006) 204–211

[4] Agrawal, R., Srikant, R.: Mining Sequential Patterns. In: Proc. 11th International Conference on Data Engineering (ICDE). (1995) 3–14

[5] Srikant, R., Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements. In: Proc. 5th International Conference Extending Database Technology (EDBT). (1996) 3–17

[6] Zaki, M.J.: SPADE: An Efficient Algorithm for Mining Frequent Sequences. Machine Learning **42**(1–2) (2001) 31–60

[7] Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. In: Proc. 20th International Conference Very Large Data Bases (VLDB). (1994) 487–499

[8] Han, J., Pei, J., Yin, Y.: Mining Frequent Patterns without Candidate Generation. In: Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD). (2000) 1–12

[9] Savary, L., Zeitouni, K.: Indexed Bit Map (IBM) for Mining Frequent Sequences. In: Proc. 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD). (2005) 659–666

[10] El-Sayed, M., Ruiz, C., Rundensteiner, E.A.: FS-Miner: Efficient and Incremental Mining of Frequent Sequence Patterns in Web Logs. In: Proc. 6th ACM Workshop on Web Information and Data Management (WIDM). (2004) 128–135

[11] de la Briandais, R.: File Searching Using Variable Length Keys. In: Proc. Western Joint Computer Conference. (1959) 295–298

[12] Aho, A.V., Corasick, M.J.: Efficient String Matching: An Aid to Bibliographic Search. Communications of the ACM **18**(6) (1975) 333–340

[13] Aho, A.V., Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms. Series in Computer Science and Information Processing. Addison-Wesley (1982)

[14] Ferreira, P.G., Azevedo, P.J.: Protein Sequence Classification Through Relevant Sequence Mining and Bayes Classifiers. In: Proc. 12th Portuguese Conference on Artificial Intelligence (EPIA). (2005) 236–247