Case Mining from Large Databases

Qiang Yang and Hong Cheng

Department of Computer Science, Hong Kong University of Science and Technology,
Clearwater Bay, Kowloon Hong Kong
{qyang, csch}@cs.ust.hk
http://www.cs.ust.hk/~qyang

Abstract. This paper presents an approach of case mining to automatically discover case bases from large datasets in order to improve both the speed and the quality of case based reasoning. Case mining constructs a case base from a large raw dataset with an objective to improve the case-base reasoning systems' efficiency and quality. Our approach starts from a raw database of objects with class attributes together with a historical database of past action sequences on these objects. The object databases can be customer records and the historical action logs can be the technical advises given to the customers to solve their problems. Our goal is to discover effective and highly representative problem descriptions associated with solution plans that accomplish their tasks. To maintain efficiency of computation, data mining methods are employed in the process of composing the case base. We motivate the application of the case mining model using a financial application example, and demonstrate the effectiveness of the model using both real and simulated datasets.

1 Introduction

To motivate the case mining problem, consider a financial application example. Suppose that a certain ABC Bank is interested in initiating a marketing campaign to encourage its customers to sign up for a new loan program. Suppose that we are given two datasets on (1) customer information and (2) past marketing action logs on customers in (1) and the final outcomes in loan-signup results. Tables 1 and 2 show an example of a customer database table together with some examples of past marketing actions on customers. Suppose that we are interested in building a campaign plan for a new customer Steve. Based on the past campaign actions for people like Steve, there are many candidate actions that one can suggestion. For example, we can use two of the past cases for John and Mary. If we follow the plan for John, we can first give him a personal phone call and then send him a gift. Alternatively, we can decrease the mortgage rate for Steve, followed by offering a new Credit card. The latter plan follows that of Mary's success. In either situation, the ABC bank might have a relatively high chance of converting Steve from a reluctant customer to a willing one.

The above problem can be formulated as a case-based reasoning problem, where the key issue is to find a good role model for customers such as Steve, and apply case

retrieval and adaptation techniques [9,10] to find low-cost plans for these customers on a case-by-case basis. Our approach is to first identify typical problem descriptions as potential role models from the customer database (Table 1). Then for each typical problem, a cost-effective plan is found to be associated with each negative-class role model (Table 2); they are stored in a case-base as a plan. Then, for each new problem, we find a similar problem description and adapt its plan in the case base.

Table 1. An example of customer database. The last attribute is the class attribute.

Customer	Salary	Cars	Mortgage	Loan
John	80K	3	None	Y
Mary	40K	1	300K	N
…	…	…	…	…
Steve	40K	1	None	N

Table 2. Action-log database example containing marketing actions and outcomes

Advisee	Action1	State1	Action2	State2
John	Phone call	Feature-values for John	Send gift	Feature-values for John
Mary	Lower mortgage	Feature-values for Mary	Credit card	Feature-values for Mary

In this paper, we focus on how to find the case base from a given database efficiently. A challenging issue facing the construction of the case base is the large size of the database and large amount of data. To solve the scale-up problem, we exploit data mining techniques, including clustering and association rule mining algorithms [2, 5, 8, 14]. Cluster center objects, also known as the medoids, are obtained from the customer database to produce the basis for cases from customers belonging to both the negative class (problem descriptions for new problems) and the positive class (where Class=Yes in the final results table). Association-rule mining is applied to the action-log database to produce typical plans to be associated with the problem-objective pairs. The resultant case base is then used for problem solving for new customers. Because we extract high-quality cases from large databases using data mining techniques, we call this method *case mining*. Empirical results are obtained to demonstrate that the case-mining algorithms are scalable and achieve a balance between quality and efficiency.

2 Related Work

Case mining is closely related to case-base maintenance, whereby the case base structure, index and knowledge representation are updated to improve problem-solving performance. The most recent special issue of *Computational Intelligence Journal* on case-based maintenance [6] highlighted this interest and progress. Smyth and Keane [15], who defined case-base competences based on how cases in an existing case base are related to each other, addressed the maintenance issue by proposing a deletion-based algorithm. Their maintenance policy is aimed at removing as much redundancy form a case base as possible by reducing the overlap between cases or groups of cases. Subsequently, Smyth and McKenna [22] and Yang and Zhu [16] explored how other policies can be applied to case-based maintenance. Leake, Kingly and Wilson [10, 11, 17] analyzed the case-base maintenance problems and cast them into a general framework of *revising* the content and structure of an existing case base.

Aamodt et. al [1] presented learning algorithms for similarity functions used in case retrieval. In our formulation, case mining is targeted for large, un-indexed raw databases rather than an existing case base. The indexing problems have been an issue of intensive study in the past, including work in [4, 19, 20]. The indexing literature assumes that there is already a case base in existence, whereas in this paper we explore how to find the case base from historical databases. Patterson et. Al [21] discussed how to apply the K-means algorithm for case-base maintenance.

The work is also related to case-based planning [9, 18], which derives a new plan based on plans stored in case bases. However, case-base planning has mainly focused on case adaptation; instead, we focus on how to exploit the distribution of massive data to obtain statistically well-balanced case bases.

3 Mining Problem Descriptions

Our algorithm is divided into two phases. In phase one, we take the customer database and find a set of typical cases by applying clustering algorithm to it. The result is a set of positive-class and negative-class objects. In order to find the typical cases, we apply two different algorithms including a clustering algorithm and an SVM-based algorithm. In phase two, we find a plan for each negative-class role model to form a potential case. Care is taken in the search process to find only high-utility plans.

We wish to find *K* typical cases to populate the case base, where the parameter K is user defined; later we discuss another method (SVM) which does not require K to be specified. These cases should be highly representative of the distribution of the customer information in the original customer database. They will serve as the initial states for our planner to work on in the next phase.

We consider all *negative* instances in the database as the training data for mining the problem descriptions of the case base. Our first method applies the clustering algorithms to the negative instances, as described in more detail in Table 3. The training database consists of the negative instances of the original database.

Table 3. **Algorithm** *Centroids-CBMine* (database DB, int K)

Steps	**Begin**
1	*casebase*= emptyset;
2	DB = *RemoveIrrelevantAttributes*(DB);
3	Separate the DB into DB+ and DB-;
4	Clusters+ = *ApplyKMeans*(*DB-, K*);
5	**for each** *cluster* in Clusters+, **do**
6	C = *findCentroid*(cluster);
7	*Insert*(C, *casebase*);
8	**end for**;
9	Return *casebase*;
	End

In the algorithm *Centroids-CBMine* in Table 3, the input database DB is a raw database. There are two classes in this database, where the negative class corresponds to population of initial states for a marketing plan. Step 2 of the algorithm performs feature extraction by applying a feature filter to the database to remove all attributes that are considered low in information content. For example, if two attributes *A1* and *A2* in the database are highly correlated, then one of them can be removed. Similarly, if an attribute *A* has very low classification power for the data as can be computed using information theory, then it can be removed as well. In our implementation, we apply a C4.5 decision-tree learning algorithm to the database DB. After a decision tree is constructed, the attributes that are not contained in the tree are removed from the database; these are the irrelevant attributes.

Step 3 of the algorithm separates the training database into two partitions, a positive-class subset and a negative-class subset. Step 4 of the algorithm performs the K-means clustering on the negative-class sub-database [5]. K-means finds *K* locally optimal center objects by repeatedly applying the *EM* algorithm on a set of data. Other good clustering algorithms can also be used here in place of K-means. Step 6 of the algorithm finds centroids of the K clusters found in the previous step. These centroids are the bases of the case base constructed thus far, and are returned to the user. Finally, Step 9 returns the case base as the output.

The centroid-based case-mining method extracts cases from the negative-class cluster center objects and takes into account only the negative class distribution. By considering the distribution of both the positive and negative class clusters, we can sometimes do better. In general, it may be better to find the boundary between the two distributions, and to select cases from the dense areas on that boundary. This is the intuition behind the SVM-based case mining algorithm.

The key issue then is to identify the cases on the *boundary* between the positive and negative cases, and select those cases as the final ones for the problem descriptions. This has two advantages. First, by using the boundary objects as cases, from the past plans, we may determine that some objects are in fact not convertible to the positive class; these are the objects that are not similar to any of the problem descriptions in the case base. The case-based reasoning algorithm cannot solve these cases. Second,

because the cases are the support vectors themselves, there is no need to specify the input parameter *K* as in the Centroids-CBMine algorithm; the parameter *K* is used to determine the number of clusters to be generated in K-means. This corresponds to parameter-less case mining, which is superior because in reality, the number of cases to be generated is typically unknown ahead of time. We observe that the cases along the boundary hyper-plane correspond to the support vectors found by an SVM classifier [7, 13]. These cases are the instances that are closest to the maximum margin hyper-plane in a hyperspace after an SVM system has discovered the classifier [7, 13].

By exploiting the above idea, we have a second problem-description mining algorithm called SVM-*CBMine()*. In the first step, we perform SVM learning on the database to locate the support vectors. Then we find the support vectors and insert them into the case base. This algorithm is illustrated in Table 4.

Table 4. **Algorithm** *SVM-CBMine* (database *DB*, int *K*)

Steps	**Begin**
1	*casebase* = empty set;
2	*Vectors* = *SVM*(*DB*);
3	**for each** *negative* support vector C- in *Vectors* **do**
4	Insert(C-, *casebase*);
5	**end for**
6	Return *casebase*;
	End

4. Mining the plans

Having obtained the problem descriptions as "seeds" for the cases in the case base, we now consider the problem of finding solution plans to be associated with these descriptions with an aim to convert all negative-class cases to positive. Our algorithm is a state-space search algorithm that search an AND-OR tree. AND-OR trees are used to deal with the problem of uncertainty in planning. The plans produced are probabilistic in nature; they are aimed at succeeding in converting the customers to positive class with low cost and high success probability.

Given the *customer* table and the *action log* database, our first task is to formulate the problem as a planning problem. In particular, we wish to find a method to map the customer records in the customer table into states using a statistical classifier. This task in itself is not trivial because it maps a large attribute space into a more concise space. When there are missing values in the database, techniques of data cleaning [14] can be applied to fill in these values.

Next, the state-action sequences in the *action log* database will be used for obtaining action definitions in a state space, such that each action is represented as a probabilistic mapping from a state to a set of states. To make the representation more realistic, we will also consider the cost of executing each action.

To summarize, from the two tables we can obtain the following information:

• $f_s(r_i) = s_j$ maps a customer record r_i to a state s_j .This function is known as the customer-state mapping function;

• $p_c(s)$ is a probability function that returns the probability that state s is in a desirable class. We call this classifier the state-classification function;

• $p(s_k \mid s_i, a_j)$ returns the transition probability that, after executing an action a_j in state s_i, one ends up in state s_k .

Our planning algorithm divides the resulting plan in stages, where each stage consists of one action and a set of possible outcome states resulting from the action. In each stage the states can be different possible states as a result of the previous action, and the action in the stage must be a same, single action. In our formulation, we define a utility function of a plan to be *u(s)* for a state *s*, where the function is defined as

$$u(s) = \max_a((\sum_{s' \in S} P(s' \mid s,a) * u(s')) - \cos t(a))$$

where, at a leaf node *t*, the function *u* is defined as the utility of the node which is *P(+|t)*, and *cost(a)* is the cost of the action *a*.

A major difficulty in solving the planning problem stems from the fact that there are potentially many states and many connections between states. This potentially large space can be reduced significantly by observing that the states and their connections are not all equal; some states and action sequences in this state-space are more significant than others because they are more frequently "traveled" by traces in the *action-log table*. This observation allows us to use an approach in which we exploit planning by abstraction.

In particular, significant state-action sequences in the state space can be discovered through a frequent string-mining algorithm [2]. We start by defining a minimum-support threshold for finding the frequent state-action sequences. Support represents the number of occurrences of a state-action sequence from the *action log database*. More formally, let *count*(*seq*) be the number of times sequence "seq" appears in the database for all customers. Then the support for sequence "seq" is defined as the frequency of the sequence. Then, a string-mining algorithm based on moving windows will mine the *action log database* to produce state-action subsequences whose support is no less than a user-defined minimum-support value. For connection purpose, we only retain substrings both beginning and ending with states, in the form of $< s_i, a_i, s_{i+1}, a_{i+1}, ..., s_n >$.

Once the frequent sequences are found, we piece together the segments of paths corresponding to the sequences to build an abstract AND-OR graph in which we will search for plans. If $< s_0, a_1, s_2 >$ and $< s_2, a_3, s_4 >$ are two segments found by the string-mining algorithm, then $< s_0, a_1, s_2, a_3, s_4 >$ is a new path in the AND-OR graph. Since each component of the AND-OR graph is guaranteed to be frequent, the AND-OR graph is a highly concise and representative state space.

Based on the above heuristic estimation methods, we can perform a best-first search in the space of plans until the termination condition is met. The termination conditions are determined by the probability or the length constraints in the problem domain. For the initial state *s*, if the expected value *E(+|s)* exceeds a predefined threshold *Success_Threshold*, i.e. the probability constraint, we consider the plan to be good enough and the search process terminates. Otherwise, one more action is attached to this plan and the new plans are inserted into the priority queue. $E(+ \mid s_i)$ is the expected state-classification probability estimating how "effective" a plan is at transfer-

ring customers from state s_i. Its calculation can be defined in the following recursive way:

$E(+ \mid s_i) = \sum p(s_k \mid s_i, a_j) * E(+ \mid s_k)$; *if* s_i is a non-terminal state; or

$E(+ \mid s_i) = P(+ \mid s_i)$ *if* s_i is a terminal state.

We also define a parameter *Max_Step* that defines the maximum length of a plan, i.e. the length constraint. We will discard a candidate plan, which is longer than the *Max_Step* but its $E(+ \mid s_i)$ value is less than the *Success_Threshold*. We now consider optimality of the algorithm.

5 Efficiency Experiments

We run tests on both artificial data sets and realistic data sets to obtain the performance of the case mining algorithm. In this paper, we are primarily interested in the speed in which to obtain the case bases, especially when the database reaches a large size. Our other ongoing work is aimed at showing the quality of the mined case bases. Thus, our interest in these tests is how the CPU time changes with different experimental parameters. In the sequel, we separately test the problem description and the planning algorithms. The experiments are performed on an Intel PC with one Gigahertz CPU.

5.1 Testing Problem Description Mining

Our first test is aimed at establishing the efficiency of the problem-description mining algorithm, which determines whether the case mining framework can be scaled up or not. It uses an artificial dataset generated on a two-dimensional space *(x, y)*, using a Gaussian distribution with different means and co-variance matrix for the positive (+) and the negative (–) classes (Figure 1). Our purpose is to demonstrate the effect of data distribution and model size on the switching-plan quality and efficiency. When the means of the two distributions are separated, we expect the class boundaries are easy to identify by the SVM-based method. As can be seen, the time it takes to build and execute the model increases with *K*, the number of cases in the case base (Figure 2). As the two distributions move close to each other such that there are no clear boundaries, the SVM-based method selects nearly all the negative examples as cases in the case base, resulting in a bloated case-base. Thus, its time expense is also very high (Figure 2). In this case, the *CenroidCaseMine* method is preferred.

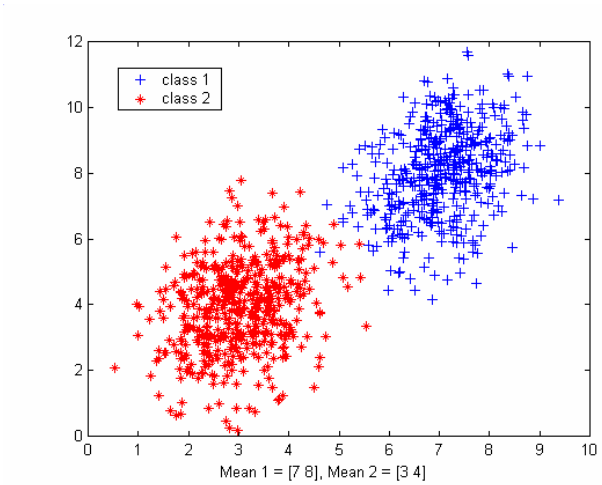

Figure 1. Distribution of the class 1=positive (+) and class 2=negative (*) data.

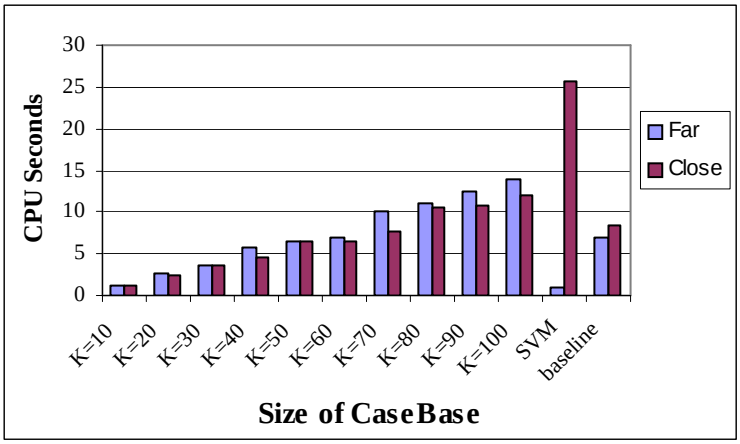

Figure 2. Comparison of CPU time for different distributions between two distributions that are either "Far" apart or Closely mixed.

For large-scale tests, we performed an experiment using the IBM QUEST synthetic data generator (http://www.almaden.ibm.com/cs/quest/syndata.html). We generated the training dataset with 9 attributes, 50% positive class and 50% negative class. An excerpt of the database is shown in Table 5.

In the dataset, the Salary attribute is uniformly distributed from 20000 to 150000, the commission values are set so that if Salary >= 75000, Commission = 0; otherwise it is uniformly distributed from 10000 to 75000. The Age attribute is uniformly distributed from 20 to 80, the Education attribute is uniformly chosen from 0 to 4, and the Car attribute includes the make of the car, and is uniformly chosen from 1 to 20.

The ZipCode attribute is uniformly chosen from 9 available zip codes, and the House Value attribute is uniformly distributed from 0.5*k*100000 to 1.5*k*100000, where 0 <= k <= 9 and the House Value depends on the ZipCode. The YearsOwned attribute is uniformly distributed from 1 to 30, and the Loan attribute is uniformly distributed from 0 to 500000. Then we compared the running time to train a 50-case case base for the centroid-based method and CPU time for the SVM based method. Our results are shown in Table 6. It is clear from the table that with large data, the centroid-based method is able to scale up much better than the SVM based method.

Table 5. An excerpt from the synthetic dataset.

Salary	Commission	Age	Education	Car	…
65498	49400	61	1	2	…
24523	0	70	2	3	…
78848	0	20	2	6	…
74340	29463	45	0	3	…
42724	0	32	1	4	…

Table 6. CPU-time comparison of Centroid-based Method and SVM-based method.

***Log*$_{10}$*(N), N*=Database size**	**CPU Time (Seconds)**	
	Centroid-based	**SVM**
2	0.7	0.7
2.5	1.6	1.9
3	6.5	14.3
3.5	23.1	319.2
4	95.8	3,834.9
4.5	312.9	No result in 5 hrs
5	1,938.4	No result in 7 hrs

5.2 Testing Problem Description Mining

In this second test, we assume that we have obtained a case base of typical problem descriptions from the negative examples in the training set. Our task now is to find a plan for each negative description. These plans will serve as the cases that can be adapted for new problems in the future. In this experiment, we test the efficiency of the algorithm for a single problem description in a case.

We again used the IBM Synthetic Generator to generate a *Customer* dataset with two classes and nine attributes. The positive class has 30,000 records representing successful customers and negative has 70,000 representing unsuccessful ones. Those 70,000 negative records are treated as starting points for *Marketing-log* data genera-

tion. We evaluated the quality of the plans via simulation. From this input, we set out to find a case base of marketing plans to convert customers to a successful class. This testing process corresponds to the testing phase for a trained model in machine learning. In this simulation, if there is a plan suitable for converting a customer record, a sequence of actions is carried out on that record. The plan will then change the customer record probabilistically. At the end, the classifier is used to decide whether the changed record has turned into a successful one.

When *Success_Threshold* is low, many states are considered positive, and thus plans can be easily and quickly found for most of the initial states in the graph. We can observe this from Figure 3: planning time is low with low *Success_Threshold*. As the *Success_Threshold* increases, so does the planning time. When *Success_Threshold* is too high, no plan can be found for some initial states. The *Time* is much higher because the searching process doesn't terminate until all the plans expanded longer than *Max_Step*. The search efficiency also depends on other parameters. *MinSupport* is another important factor. When *MinSupport* is low, more plans in the action log qualify to be in the abstract state space, and thus search takes a longer time. We can also see from Figure 4 that the CPU time drop is greater when the MinSupport increases from one to 100. However, the planning time drop slows down between MinSupport value of 100 and 1000. This suggest to us that we should use a support value of around 100, since at this value we can achieve the balance of a relatively large number of plans to be included in the search space and a low planning time.

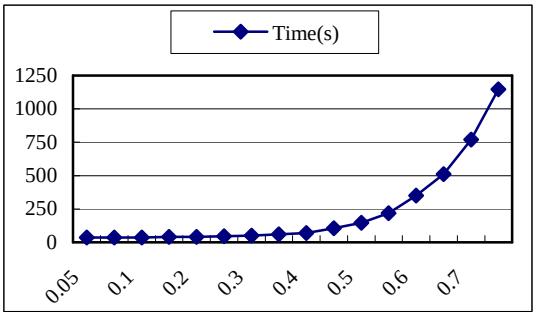

Figure 3. CPU Time vs. Success_Threshold. MinSupport = 100

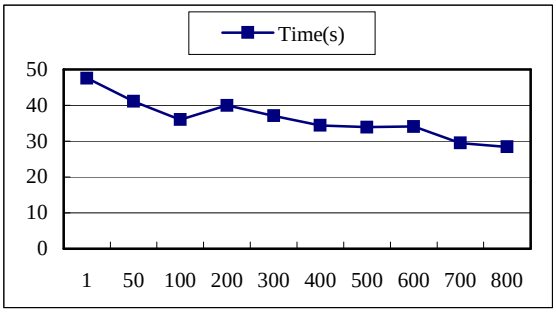

Figure 4. CPU Time vs. *MinSupport*. Success_Threshold=0.05.

6 Conclusions and Future Work

In this paper, we presented a case mining algorithm for discovering a case base from a large database. The central issue of the problem lies in the discovery of high-quality case bases that balances the efficiency for extracting the case base, the cost of the adaptation plans and the probability of success. This is what we call the case-mining problem. For the data distribution where the two classes are clearly separated, the SVM-CBMine algorithm, which is an SVM-based method, should be used. When the data distributions are not separated well by a boundary, the cluster-centroids based method is recommended. Furthermore, we designed a planning system for finding probabilistic plans where the probability of success is taken into account. The efficiency of the case mining algorithm is validated against large databases. In the future, we will continue to explore quality of case mining results and the related problem of case adaptation in this framework.

References

[1] A. Aamodt, H. A. Sandtorv, O. M. Winnem: Combining Case Based Reasoning and Data Mining - A way of revealing and reusing RAMS experience. In Lydersen, Hansen, Sandtorv (eds.), *Safety and Reliability; Proceedings of ESREL '98*, Trondheim, June 16-19, 1998. Balkena, Rotterdam, 1998.

[2] R. Agrawal and R. Srikant. 1994. *Fast algorithm for mining association rules*. Proceedings of the Twentieth International Conference on Very Large Databases. pp 487-499

[3] C. L. Blake, C.J. Merz (1998). *UCI Repository of machine learning databases* Irvine, CA: University of California, Department of Information and Computer Science. http://www.ics.uci.edu/~mlearn/MLRepository.html

[4] A. Bonzano, P. Cunningham and B. Smyth *Using Introspective Learning to Improve Retrieval in CBR: A Case Study in Air Traffic Contol*, in Leake D., Plaza E. (eds.) `Case-Based Reasoning Research and Development, Second International Conference on Case-Based Reasoning ICCBR 1997, Springer Verlag, Berlin 1997.

[5] P. S. Bradley and U. M. Fayyad. *Refining initial points for **k-means** clustering*. In Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98), pages 91--99, San Francisco, CA, 1998. Morgan Kaufmann.

[6] *Computational Intelligence Journal, Special Issue on Case-base Maintenance*. Blackwell Publishers, Boston MA UK. Vol. 17, No. 2, May 2001. Editors: D. Leake, B. Smyth, D. Wilson and Q. Yang.

[7] G. C. Cowley. *MATLAB Support Vector Machine Toolbox. v0.54B* University of East Anglia, School of Information Systems, Norwich, Norfolk, U.K. NR4 7TJ, 2000. http://theoval.sys.uea.ac.uk/~gcc/svm/toolbox

[8] P. Domingos and M. Richardson. Mining the Network Value of Customers. Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. August 2001. ACM. N.Y. N.Y. USA

[9] K. Hammond. Explaining and Repairing Plans that Fail. *Artificial Intelligence*, 45(1-2):173-- 228, 1990.

[10] D. Leake. *Case-based Reasoning -- Experiences*, Lessons and Future Directions. AAAI Press/ The MIT Press, 1996.

[11] D. Leake, Kinley, A., & Wilson, D. (1995). Learning to improve case adaptation by introspective reasoning and CBR. In Proceedings of First International Conference on Case-Based Reasoning. Sesimbra, Portugal.

[12] C. X. Ling and C. Li. *Data mining for direct marketing: Problems and solutions.* In Proceedings 4th International Conference on Knowledge Discovery in Databases (KDD-98), New York, 1998.

[13] J.C. Platt, *Fast training of support vector machines using sequential minimal optimization*, in Advances in Kernel Methods - Support Vector Learning, (Eds) B. Scholkopf, C. Burges, and A. J. Smola, MIT Press, Cambridge, Massachusetts, chapter 12, pp 185-208, 1999.

[14] J. Quinlan C4.5: *Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., San Mateo, CA.

[15] B. Smyth and M. T. Keane. *Remembering to forget: A competence--preserving deletion policy for case--based reasoning systems*. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp 377--382, 1995

[16] Q. Yang and J. Zhu. A Case-addition Policy for Case-based Reasoning. *Computational Intelligence Journal*, Special Issue on Case-based Maintenance. Blackwell Publishers, Boston MA UK. Vol. 17, No. 2, May 2001. Pages 250--262.

[17] D. C. Wilson and D. B. Leake *Maintaining Case-based Reasoners: Dimensions and Directions.* Computational Intelligence Journal. Vol. 17, No. 2. May 2001

[18] M. Veloso. (1994). *Planning and learning by analogical reasoning*. Number 886 in Lecture Notes in Artificial Intelligence. Springer Verlag.

[19] D. Wettschereck and D.V. Aha. Weighting features. In Proceedings of the First International Conference on Case-Based Reasoning, ICCBR-95, pages 347--358, Lisbon, Portugal, 1995. Springer-Verlag.

[20] Zhong Zhang and Qiang Yang. Feature Weight Maintenance in Case Bases Using Introspective Learning. *Journal of Intelligent Information Systems*, Kluwer Academic Publishers, 16, Pages 95--116, 2001. The Netherlands.

[21] D. Patterson, N. Rooney, M. Galushka, S. S. Anand: Towards Dynamic Maintenance of Retrieval Knowledge in CBR. In Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference, 2002, Florida, USA. AAAI Press 2002, pages 126-131.

[22] B. Smyth, and E. McKenna, Building Compact Competent Case-Bases. In *Proceedings of the third International Conference on Case-based Reasoning*, Springer-Verlag, Munich, Germany, 1999, pp 329-342.