Using Genetic Algorithms to Discover Selection Criteria for Resolving Contradictory Solutions Returned by CBR

By

Brent Stephens

B.S.C.S. University of Kansas, Lawrence, Kansas, 2000

Submitted to the Department of Electrical Engineering and Computer Science and the Faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

_____
Professor in charge

_____

_____
Committee Members

_____
Date thesis accepted

Acknowledgements

I would like to thank Dr. Costas Tsatsoulis for his many years of patience and encouragement throughout the creation of this thesis. I would also like to thank Dr. Leen-Kiat Soh, Heather Amthauer, and Todd Blackmon for their help throughout the process. Also, I would like to thank Dr. Susan Gauch and Dr. Arvin Agah for agreeing to be on my committee.

Finally, I would like to thank my family and friends for the years of support and occasional open derision about finishing my thesis. I would especially like to thank Tom Conroy for providing me with the quality food at a reasonable price that sustained me through some long typing sessions.

Abstract

Case-Based Reasoning (CBR) is a useful method for performing classification in the corporate environment. Traditional forms of matching and case selection in CBR do not perform well when faced with noisy, outdated, or cyclical data mixed into the case base. By training Genetic Algorithms (GAs) to differentiate between cases based on secondary properties of the cases selected by the CBR system, a combined system may be able to improve on traditional CBR selection methods. The goal of this thesis is to evaluate the performance of such a system using a real-world, corporate database, and assess its effect on the accuracy and coverage of the system. Our domain is the assignment of billing codes to shipping statements, and the case base contains numerous overlapping or identical cases. In this domain, cases contain contradictory or incorrect information due to problems in the domain that are not included in case descriptions. In assigning billing codes, we assumed that the frequency with which a billing code appears in similar cases, as well as the recency with which appeared were indicators of the significance of a case. As there was no specific method for applying these measures, GAs were used to determine appropriate ways to use them. GAs allowed us to discover methods to implement the frequency and recency measures in a way that improved the accuracy and coverage of the CBR system.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Case-Based Reasoning (CBR) is a method for using previous experiences to solve new problems while reducing the need for user decision making. This previous experience is stored in the form of problem-solution pairs, or cases. For CBR to be effective, cases with similar problems need to have similar solutions and the data provided to it needs to be accurate. When these assumptions do not hold, the program needs additional ways to measure the similarity of cases or select a solution. The ability of users to analyze the domain and improve the selection process is often limited due to complexities in the data, and expert knowledge is prone to errors and misassumptions. Machine learning can be used to improve the process, however. A Genetic Algorithm (GA) is a machine learning method that simulates an evolutionary process. Through experimentation, we will determine if combining GA and CBR will provide a way to reduce the reliance on user knowledge and improve selection accuracy.

CBR has been used for selecting appropriate solutions in many diverse applications. One potential field of interest is in correcting form errors that occur when dealing with billing codes for a shipping company. One of the most common errors for these companies is incorrectly assigning billing information. These errors occur when a clerical error is made, outdated information is used, poor decision-making occurs, or

from a variety of other causes with respect to the data used to assign a billing code. The results of the errors are a bill with an incorrect amount, a bill for the wrong item, sending a bill to an incorrect customer or office, or being unable to determine who to send the bill to at all. The effect on the shipping company is that the billing process takes longer and the company must pay workers to spend hours manually correcting the data. Attempts have been made to encode a rule set for correcting information in the billing process, but the results have been large, complex, and difficult to maintain. CBR has the ability to automate this process, saving a company both time and money. Being able to determine who to bill, rather than assigning an incorrect price or code, is a useful problem to resolve, since it is the most common problem, and can be determined before an initial, incorrect billing occurs. This is the problem that we focused on.

The case base that we obtained for testing was from Burlington Northern Santa Fe Railroad (BNSF). The case base is composed entirely of cases that had to be manually corrected. Each case consists of 20 attributes that were selected as relevant to the matching process, a timestamp, and a billing code. The cases had no billing code assigned to them initially due to errors in attribute values or changes in the domain that were not properly dealt with. The solutions for each case have been corrected, so the case set can be used for testing and training.

The data set produced by BNSF contains many nearly identical shipments in terms of sender, recipient, product, and other features. Some errors in the billing process result from small errors that do not affect the correctness of the assigned billing code. When these cases are retrieved as possible solutions, the differences do not affect the CBR's ability to assign the correct billing code without any additional assistance. New cases are added to the set of available cases after each successful transaction and provide updated data for when shipping information changes. This may add noisy data to the case set, however, and needs to be taken into account when assigning billing codes.

The difficulties in this particular domain come when we must select between anomalous events that break CBR assumptions. To help understand these anomalies, we can break them down into three categories:

1. Shifts in billing practices, in which old cases are superceded and need to be replaced by a newer case.
2. Cycles in billing practices, where seasonal changes occur and the correct case shifts between two or more solutions at regular intervals.
3. Data errors, which often result in cases with different shipping codes being similar enough to the current problem so that any of them may be considered a match.

When competing solutions are retrieved by the CBR system, there is no established method to select between them.

Maintenance on the case base can be used to eliminate some of these problems. A large amount of work has been done in periodic maintenance of case sets that help reduce redundancy and eliminate noisy and incorrect data. The goal of this maintenance is to reduce the size of the case set and eliminate any inconsistencies. Maintenance of this type reduces the case base to only the cases that are correct at the time it is performed, and then eliminates as many redundancies as possible from that set. This technique is not applicable in our domain.

The redundancies in the case set represent the best practice in the company, and inconsistencies frequently represent seasonal changes in shipping information. What is needed instead is a system that can adapt quickly to a paradigm shift without losing previous knowledge. To do this, we use all available cases when doing retrieval, and then select the best case or set of cases from all retrieved similar cases. When there are conflicting solutions in the resulting set, conflict resolution must be done, either by selecting from possible cases or stating that no decision can be made. Two common approaches to conflict resolution are to choose either the most frequently occurring solution or the most recently occurring one. Simply using frequency or recency limits accuracy in our domain, because it is not well understood how to apply these measures.

To overcome this, we select the best solution by creating a set of rules that apply these two properties, and then combine the resulting scores. Since the nature and

form of these rules is not known for our domain, we used basic equations for assessing the importance of the features, and used a GA to find optimal values for the equations. A second GA is then applied to find the relative importance of each function in selecting a solution.

The equations used in assessing frequency and recency create a score based on either the percentage of cases with a given solution within the set of retrieved cases or a score associated with how recently the case occurred. This value is combined with the similarity of the retrieved case to the current case to generate a score. The scores for all cases with a given solution are totaled to give a final score for that solution. This is repeated for each possible solution retrieved. This process is repeated for each equation. Once a score for each solution has been generated by each equation, the scores are weighted and combined. The solution with the highest total score is selected by the program. The goal of this project is to maximize the percentage of cases which we could correctly classify. When using the lowest similarity testing in this domain, 57% of input cases returned conflicting solutions from the initial matching. Applying this methodology, we were able to correctly classify 48% of the input cases, or 80% of the cases with conflicting solutions. Simpler approaches to selecting a solution, such as using the most recent case, the most frequently occurring solution, or using a K-Nearest Neighbor solution, were only able to correctly classify 23% to 63% of cases that returned conflicting solutions.

Chapter 2

Problem Definition

This thesis proposes a solution to the problem of selecting between competing solutions returned by a case-based reasoning (CBR) system. If similar problems in the case base have very different solutions, a CBR system needs a way of determining which of the conflicting solutions best fits the current problem. This problem is acute when the case base is a large corporate database, since such databases tend to contain errors, outdated historical information, and so on. In this chapter we discuss why CBR can be used to identify solutions in a business environment, how conflicting solutions can be retrieved, and why existing research does not address the problem of automatically selecting a solution amongst a retrieved set of conflicting ones.

Databases produced by corporations contain large amounts of data. One possible use of the information is classification. To perform classification, the description of an old situation is matched to a new situation and the classification of the old is applied to the new one. Simple classification by matching cases requires that identical problem descriptions will occur and that solutions from previous experience can be applied. The only requirement for simple matching is that identical problems must have identical solutions. When descriptions can be matched exactly and solutions do

not change, classification can be done by simply querying a database. When these assumptions do not hold, simple classification by querying cannot be done.

Case-Based Reasoning (CBR) is a method for using previous experience to solve new problems with minimal user intervention. CBR can be used for classification when using real world databases because it is capable of selecting the best approximate match when there is no exact match. In the simplest form of CBR classification, a new case is matched against the cases in the case base. The classification of the most similar case from the case base is used as the classification of the new case. After a case has been classified, it can be retained in the case base to expand coverage, because the case is not necessarily identical to existing ones [Aamodt and Plaza, 1994]. Cases can also be added manually.

The difficulties in applying CBR to corporate databases are that these databases are often dynamic, inconsistent, and very large. New cases added to the database may be similar to older cases and have different solutions. This can happen when either the correct solution changes or when the new case is incorrect in some way. Many methods have been developed to select appropriate cases from a case base. There are several forms of heuristics that are applied to the case descriptions when performing matching to select between competing solutions. The specific heuristic needed is dependant on data representation and domain specific knowledge. When heuristics fail to identify a single case to apply, either because no case can be identified as

similar or multiple equally similar cases are selected, intervention is needed to select the appropriate case. Automatically selecting between solutions/classifications when equally similar cases exist has not been addressed. Most CBR systems either require human intervention at this point or simply leave the problem as unsolved.

A CBR program capable of using business databases, must address the problems described above. Specifically, for CBR to use business databases the program must be able to select between conflicting results which may be returned from the similarity-assessment component of CBR. CBR may return contradictory solutions because the descriptions included in each case often do not provide enough information to distinguish between two different situations. In spite of this, a CBR program must be able to select the correct solution for a given situation as often as possible. A second reason why conflicting solutions are returned is simply due to errors and noise in the database. When new cases are added that have the same or very similar problem description as existing cases but have different results, the program must be able to recognize which new cases are noisy and which accurately represent a legal, valid change in the domain. Since the cases are similar according to the normal matching procedure, features that are not used initially may need to be considered. The features used to differentiate noisy cases from domain changes may come from an individual case or may be collected from a set of cases. A third reason for conflicting solutions is that often there are changes in the domain and the correct solution for two identical problems has changed. CBR assumes that at any given

time, all the cases in the case base contain correct solutions to problem definitions. Cases which are inaccurate are assumed to have no value. This assumption does not necessarily hold when dealing with real world data. Frequent additions to a database may introduce conflicting solutions, but both solutions may have been correct when they were entered, and may be correct at some time in the future. Eliminating cases because they are not accurate at one given moment may reduce the ability of the CBR program to accurately classify cases over time.

One proposed solution to the retrieval of contradictory solutions by a CBR system is to assume that any contradictions in the case base are due to errors, and are logically inconsistent. To prevent conflicting cases from being returned by the CBR, consistency maintenance may be performed on the case base. To perform consistency maintenance, cases are either clustered by problem description, where similarly described problems with different solutions are inconsistencies that need to be eliminated, or are clustered by solution and generalized cases are created that reduce conflicts. This can be performed either off-line, on the case base as a whole, or on-line when new cases are added to the case base, and there are a variety of methods to be used [Surma and Tyburcy, 1998]. Consistency maintenance in this situation either requires user interaction to eliminate conflicts or requires additional domain knowledge for resolving conflicts.

The problem with consistency maintenance is that, first, it ignores history (companies may want to keep old cases in their databases, even though their current solutions have changed), second, it ignores business cycles (some solutions may change cyclically), and, third, it ignores the fact that some competing solutions may be unique or one-time but still useful. The domain knowledge needed to address these problems is often difficult to formalize and final selection may need to be performed by a human. Over generalization of the case base may also reduce the problem to a rule based system, completely eliminating the benefits CBR. What is needed is a methodology that can select amongst competing solutions without first requiring that all such solutions be eliminated from the case base, and which also can detect conflicts due to changes in the domain versus transcription errors and noise. In our work we propose such a solution.

The second type of case base maintenance tries to remove or summarize redundant cases (i.e. similar cases with similar solutions), so as to reduce computational complexity and make long-term manual maintenance easier. The claim is that for a CBR application to be useful over an extended period of time in a dynamic domain, maintenance is needed to keep the case base up to date with new concepts. When the rate of change for cases in the case base becomes very high, it is difficult to manually maintain the case base. Also, since these case bases are generally large, maintenance is a computationally complex problem. As a result, any solution must be computationally minimal and minimize human intervention. When databases are

large, frequent maintenance may require more human effort than is saved by using CBR. To reduce the complexity, redundant cases can be eliminated from the case base. This is done by grouping cases by solution, then generalizing the problem descriptions to cover a subset of similar cases [Racine and Yang, 1997]. A problem with applying this type of maintenance is that it requires a case to be represented and matched in a more abstract way than is needed when directly using a database. This requires additional knowledge of the domain. A second problem is that the case base loses richness of data, which may not be explicitly stored in any one case. In our research we propose a way of actually utilizing the case base redundancy to select between competing solutions. Our work saves time, since maintenance is computationally expensive and may require human interaction. Our method is sufficiently robust to compensate for changes in the case base without the need for continuous maintenance.

Chapter 3

Related Works

Case Based Reasoning (CBR) is a reasoning method that draws information from specific prior situations, rather than from generalized rules [Leake, 1996]. The application of specific situations for reasoning in CBR is modeled on learning theories that believe that much human understanding is based on the application of specific instances from experience [Schank, 1982]. Using this model, CBR systems reason about new situations based on previous experience, then learn by adding the new situations along with their solutions to their knowledge bases. CBR is often used for emulating or assisting human decision making because of these perceived similarities between it and human reasoning.

CBR implementations have been divided into three major models:

1. Task-Based, where data representation and selection models are highly specific
2. Enterprise, where data representation is limited and selection models are generalized and adapted to the data.
3. Web-based, were XML or similar representations are used so that data can be represented in complex ways and selection models can still be general [Sengutpa, Wilson, and Leake, 1999].

Task-Based implementations develop the data representation and selection method in tandem. They tend to be easier to create since the developer has a greater degree of freedom. It is difficult to adapt these solutions to other domains, however, because the two parts are heavily intertwined. Enterprise implementations develop a selection method to conform to existing data. The existing data generally does not have the complexity that it would in Task-Based implementations, because it was generally not created with CBR in mind. The benefit of Enterprise implementations is that data representations across domains tend to be similar enough that adapting solution methods is easier. Web-based implementations have many of the benefits of both. XML representations allow for the complexity of data representation of a Task-Based implementation, but are sufficiently similar in structure to allow solutions methods to be readily adapted between domains. The limitation on developing Web-Based implementations is that transferring Enterprise data representations to Web-Based adds nothing to the complexity of the representation and Web-Based are simply an additional burden to the development of Task-Based solutions.

Business databases contain large collections of information to which CBR can potentially be applied. In this domain, CBR can automate decision processes and help businesses capture human understanding of a process that may only be known to a few individuals. There are many difficulties in using CBR in these settings, however. Databases often contain erroneous or redundant data and are generally not organized in a way that is efficient for CBR. Large databases, common in business

domains, are difficult to efficiently process. There are several methods for reducing their size, but this is a tradeoff because either accuracy or coverage will be diminished [Smyth and Cunningham, 1996]. Also, experts in the domain often have difficulty in formalizing their decision making process or there may be no experts for the domain, known as the "knowledge elicitation bottleneck" [Watson, 1997]. Last, if the CBR processes a large number of cases and adds the new cases to its knowledge base, the system must maintain accuracy and usability either by adapting the selection methods or the knowledge base [Heister and Wilke, 1998].

Case-Based Reasoning

Case-Based Reasoning centers around the idea that experiences tend to recur and that previous solutions can be reapplied to similar, new situations. New experiences are added to the system when they are new experiences or could improve future solutions. CBR systems become more competent over time as they are able to solve a greater breadth of problems [Kolodner, 1993]. The Case-Based Reasoning process has been explained as a four step process:

1. Retrieve similar cases.
2. Reuse information or knowledge from previous situations.
3. Revise and adapt the solution to the current situation.
4. Retain useful information for learning [Aadmodt and Plaza, 1994].

Not all CBR systems apply these steps equally, however. As we will discuss later, Case-Based Learning systems, a class of CBR, do not perform any revision or adaptation to cases.

Retrieval of similar cases can be performed in a variety of ways. One basic method, and the one used for the initial matching step of the program described in this thesis, is a variation on the nearest neighbor algorithm [Globig and Wess, 1995]. In the nearest neighbor algorithm, attribute values are simply compared between the new situation and the one in the case base that we are comparing it to. We will refer to the new situation as *N* and the current case *C*. Attributes for each case are referred to by a subscript *i*. For each attribute, there is a function, *f*, that assigns a score of between 0 and 1, depending on the relationship of the two values. The similarity score for two cases is the summation of the functions over all attributes. Cases are ranked according to score, the highest scoring cases being the most similar. Cases are then applied in the order of similarity [Kolodner, 1993].

The similarity metric that we used made three modifications on this algorithm. First, each attribute was assigned a weight, which modifies the result of *fi* when computing the final score. Second, some attributes were determined to be mandatory for two cases to be considered matching, so an overall score of 0 was assigned if these attributes did not match according to *fi*. Lastly, scores were normalized to between 1 and 0, to simplify later processing. Since we treated all attributes as symbols that we

did not interpret, *f* is always simply whether or not the values of the two attributes are equal.

With respect to learning, CBR systems can be separated into four separate knowledge "containers":

1. How cases are stored or represented.
2. The similarity measure.
3. The cases in the case base.
4. The adaptation and revision methods [Richter, 1995].

The containers vary in importance from system to system. Likewise, the competency of a system in one of the containers can overcome deficiencies in other containers. Adding additional expert knowledge or applying machine learning to a container represents a separate opportunity to improve the overall the performance of the system. A framework for performing knowledge acquisition to the CBR process and applying this information to these containers was described Wilke et al. [1997]

In most CBR systems, the similarity metric distinguishes the most appropriate case. When no single case distinguishes itself or no case is sufficiently similar, then the solution is left to the user. The solution is generally presented to the user either as the solution, the solution and the score, or the results of a simulated application of the solution [Cheetham, 2000].

Some systems have dealt with the problem of case selection by developing secondary metrics to present to the user, as well. One such system developed a confidence measure by clustering potential solutions and assigning a solution to a quality category based on its score and the number of similar cases with similar solutions [Cheetham, 2000]. This process is similar to clustering the final results to provide a more nuanced score for the quality of a solution. This is similar to our method of developing secondary similarity metrics to improve system accuracy. In our system, however, the clustering occurs by default because solutions are repeated numerous times throughout the case base. In addition, our system looks at additional features that are not used in the base similarity metric.

The assumption that cases which are similar will have similar solutions holds, even when new cases are added to the case base, by performing maintenance to keep the case base consistent and to eliminate cases that are identical or overly similar to existing ones [Racine and Yang, 1997]. To integrate the maintenance process into CBR, two additional steps can be added to the model described by Aadmodt and Plaza [1994]:

5. Review the current performance of the system.
6. Restore the system to the best possible condition through maintenance [Roth-Berghover and Iglezakis, 2001]

One of the major advantages of performing maintenance is an improvement in retrieval times and system performance. CBR has the advantage of being a lazy

learning system, so that it "should not be possible for system competence to decrease with an increase in case-base size." [Smyth and Cunningham, 1996]. This assumption is important to our system, as maintenance tends to be an expensive operation that would not fit in our domain.

The problem of dealing with concept drift and noisy data has been well documented. The FLORA4 system dealt with this problem by maintaining generalized solutions to which a confidence level is assigned. Confidence levels are increased when a solution is correctly assigned and decreased when it is incorrectly assigned. When dealing with noisy data, confidence levels may periodically drop, but when the confidence level drops below a threshold level, it is taken as a sign of concept drift, and the solution is no longer used [Widmer, 1994]. Aha also found some success in dealing with noisy cases by updating similarity metrics and forgetting of cases that misclassify, but did not deal with time varying domains [Aha, 1991].

The knowledge container in our system that best lends itself to maintenance is the case selection process. This has been found to be a valuable method for adjusting a CBR system to changing domain conditions [Wilson, 2001]. Most of the research in this area has centered on adjusting the weights used in the similarity measure. In most systems, features weights are created and adjusted manually, making for a process that is "highly informal and inaccurate, but also involving intensive labor." [Zhang and Yang, 2001]. These conditions describe our system as well, but the

weights used in the similarity metric are not always the source of conflicting cases. This solution does not address conflicts resulting from shifts in the domain, from errors in the input case, or errors that are in the case base.

One problem that CBR has been used to address is classification. Classification is much like any other reasoning task performed with CBR, only the implementations tend to not perform any adaptation and have simpler case representations. CBR classifiers are referred to as Case-Based Learning (CBL) systems [Aha, 1991]. The problem of learning with CBL is limited to two options: modifying the case base and modifying the similarity measure. Adding new cases to the case base is necessary for the system to learn new concepts, as no adaptation is done. The similarity measure is then adapted to improve the accuracy of the system over known concepts [Globig and Wess, 1995].

One implementation of a CBL system is PROTOS. PROTOS uses surface features to determine the category to which a new situation belongs then selects and returns the most appropriate case. Learning in PROTOS is supervised, and is done by restructuring the case base after failed classification [Bareiss, Porter, Weir, 1988].

Genetic Algorithms

Genetic algorithms (GAs) are a search method based loosely on evolutionary patterns in nature. GAs can be used to quickly find near optimal solutions in domains too

complex for exhaustive search [Holland, 1975]. GAs encode possible solutions, *individuals*, into bit strings or other simple representation. A population of individuals is maintained and evaluated in parallel. For each generation, the fitness of an individual is evaluated, which determines its likelihood to continue to influence future generations [Liu, 1996]. To apply this model, there are three requirements for the system:

1. A "Darwinian" fitness evaluation
2. A "Mating Operator" to select individuals from which to produce the next generation
3. "Genetic Operators" to determine composition of new individuals from parents [De Jong, 1988].

One basic use of GAs is for parameter optimization. Holland described a simple implementation of this method using a GA to assign weights to a simple pattern recognizer, among other possible uses [1975]. De Jong also described the ability of GAs to efficiently search for parameters using these methods [1988]. There has been extensive further research into applications and optimizations for GAs. As such, we will limit the discussion to use of GAs with CBR.

Much research has been done in integrating GAs and CBR. Most of this work falls into four categories:

1. Using CBR to seed the GAs initial condition

2. Using GAs to perform adaptation.
3. Using GAs to create a case base
4. Using GAs to improve the similarity metric

When using CBR to seed the initial GA conditions, the premise is that the evolutionary process can be expedited by beginning the process with some individuals of known quality, rather than completely random. One example was comparing the performance of Case Initialized GAs (CIGA) to Randomly Initialized GAs (RIGA) in combinatorial circuit design [Liu, 1996]. It was found that setting 5-20% of the initial GA population to cases stored from similar, previous GA runs could improve the fitness of the resultant solution. Similarly, GAs have been used in the SAMUEL system to learn multi-agent strategies in changing environments [Ramsey and Grefenstette, 1993]. In this system, GAs were used to adapt strategies learned in a static environment to changes that have been detected in the environment. In this system, the GA was initialized with the previous strategies from the case base that most closely resemble the new situation.

Several systems also use GAs to adapt the cases retrieved by a CBR system. Conceptually, these systems are very similar to those that use CBR to initialize the GA. In the Case Initialized Genetic AlgoRithm system, (CIGAR), for example, 10-15% of the population for each generation is injected cases from the case base [Louis and Johnson, 1999]. Cases are stored in the case base from various times in the evolutionary process, rather than only at the end, and cases of varying fitness, rather

than only the fittest solutions, are injected to help maintain a diverse population. This method helps the system avoid local optima, and tends to reach a more optimal final solution. In this system, the evolution process is paused at various times and individuals from the population are clustered according to fitness and genetic makeup. Prototypes from these clusters are then added to the case base and periodically re-injected into the evolution process or reused in future GAs [Perez et al, 2002].

Applying GAs for adaptation in a CBR system that performed constraint satisfaction in building design was described by Maher and de Silva Garza [1996]. In this system, initial designs retrieved from a case base were used to seed the entire GA population, and then a GA was used to find combinations of the two designs that satisfied the maximum number of constraints for a new design. The design and lessons learned from previous solutions could then be applied to minimize the amount of work repeated in completing the new design. A similar system expanded on this idea, only expanding the concept of adding cases from various points in the evolutionary process. A similar constraint satisfaction problem was addressed by Hunt [1995]. This system also applies adaptation rules similar to traditional CBR to individuals at each step in the evolutionary process to keep individuals within legal limits. The COMPOSER system also used GAs to perform constraint satisfaction [Purvis and Athalye, 1997]. COMPOSER differed slightly in that the end target of

the GA was the optimal combination of several competing cases, were as other systems used competing cases as a starting point in the search for an optimal solution.

Some work has also been done in using GAs to create a case base when none initially exists [Soh and Tsatsoulis, 2001]. In this work, an evaluation criterion exists for members of the GA population. As cases of sufficient quality are created by the GA, they are added to the case base for future use. Maintenance of the case base takes place when there are sufficient cases stored and sufficient time to perform the operation, and is done by applying similar genetic learning.

A small amount of other work has been done on performing maintenance on a case base using GAs. In this work, cases use GAs to search "for the best location in the case space and for the best weight system associated." [Huang, 1996]. In this system, rather than using globally defined GAs, agents use GAs to find local optima, and agents interact to attempt to locate the globally optimal solution. A hill climbing approach to GAs has also been used to cluster cases in a case base and speed up nearest-neighbor searches [Skalak, 1994]. In this implementation, clustered cases were encoded and fed into a GA, and the resulting prototype case that was best at classifying the cluster was used in place of the group of cases. To further reduce the search, a GA was applied to select the optimal features to search over. By applying GAs in this way, search times were greatly reduced without sacrificing accuracy.

A large amount of work has been done using GAs to optimize the similarity metric. Kelly and Davis used GAs to find feature weights in a k-nearest neighbor matching system [1991]. A CBR system applied to the Vibration Case Library, which is used to diagnose machine faults, applied a GA to optimize feature weights used in similarity matching [Oatley et al, 1998]. In this system, the ranked order of cases returned by CBR are compared to the order given by a domain expert to determine the fitness of an individual from the GA. Similarly, GAs have been applied to find optimal weights in finding similar Corporate Bond Ratings [Shin and Han, 1999]. This system compared the results of nearest neighbor matching using feature weights given by domain experts, feature weights derived using a GA, and combining the two with various level of significance given to each. They found that in their domain, the system using only the weights from the GA outperformed all the systems using experts' weights. Kool et al. applied GAs to feature selection and feature weighting in natural language processing [2000]. They found that the GAs greatly outperformed unweighted versions of their system, and performed favorably compared to decision tree and information gain based learning methods.

Chapter 4

Domain Description

The problem that we addressed using CBR was correcting billing statements, *waybills*, for Burlington Northern Santa Fe Railway (BNSF). Each billing statement contains over 700 pieces of information about the source and destination location, sender, receiver, product being shipped, method of shipment, and other domain related information. Of the 700 fields, 20 were selected as being potentially interesting to the matching process. The information on a waybill is used to determine a billing code. For an example of a waybill, please refer to Appendix A. Shipping codes are unique to a particular billing rate and billing address for a company, but many other fields from the waybill affect billing code assignment. When relevant data from the waybill and billing information is identical, the same shipping code is used, but when a waybill or billing data varies, different billing codes are used.

Billing codes are normally assigned by BNSF employees. When there is conflicting information contained within the waybill, no billing code is assigned. The waybill is then assessed to find the source of the conflict, corrections are made, and a shipping code is assigned. Often, billing codes are assigned incorrectly either due to human error or mistakes on waybills. These errors are often not detected until later in the billing process or are only identified when a customer rejects or does not receive a

bill. Waybills that need to be corrected at this point are returned to BNSF where changes are made and the client is rebilled. In both scenarios, any changes made to the waybill or billing code are tracked to provide a detailed record of the correction process.

BNSF records show that about 5000 billing statements need to be corrected per month out of about 600,000 bills paid. Manually correcting the bills is a time consuming and monotonous task requiring several individuals working full time to perform. Manually correcting billing codes also slows the receipt of payment from customers. Attempts have been made to generate a set of rules to automate this process, but the rules tend to quickly become outdated and are difficult to maintain.

The database of closed out waybills stores all shipping records along with a record of changes that are made. The final waybills are nearly all correct, although some incorrect data on waybills or misassigned billing codes may make it through the entire billing process without being corrected. A large percentage of the shipments made by BNSF are the same as a previous shipment, differing only in when they were shipped and a few other fields, none of which affect shipping code assignment. New waybills can frequently be compared to older waybills to find the correct billing code assignment. Since the database is maintained over a long period of time, many billing codes eventually expire and are replaced with new ones. Also, some billing codes are seasonal, and older codes may replace newer ones at different times. Lastly, some

billing codes are only used temporarily, due to changes in billing address or a temporary agreement. In this situation, older codes may replace newer ones at an irregular interval.

The overall goal of this project was to assign correct billing codes to waybills that had no code assigned initially or when it was recognized that the billing code was incorrect at some point in the billing process. The challenge in doing this is that there are errors on new waybills and that, even with correct waybills, billing codes cannot be blindly assigned based on old records. An additional challenge is differentiating between a billing code that was incorrectly assigned, due to either a code simply being entered wrong or a new billing code superseding the one assigned, and incorrect waybill information leading to the assignment of an incorrect code.

The particular situation which we addressed was to assign billing codes when no code was initially assigned. Discussions with BNSF's shipping experts found that there are a limited number of fields that are assessed when selecting a billing code. Several of these fields needed to be exact matches for two shipments to have the same billing codes, and the rest were assigned weights according to their importance in the matching process.

When correcting billing codes with a CBR system using these criteria, it was found that often there is only one billing code on any similar shipments. In these cases, a

billing code could be assigned without further analysis. More difficult cases were when multiple billing codes were returned by CBR. Further discussions found that two criteria used to select a billing code in this situation are the *frequency* with which a previous code had been used and the *recency* with which a code had been assigned. Frequency refers to the volume of previous similar waybills that had been assigned a particular code and recency refers to the most recently assigned code for a similar waybill.

There are no set rules on how to apply recency and frequency in assigning codes. Cyclical billing patterns, outdated billing codes, and errors in data all need to be taken into account when trying to apply them. Just because a billing code was applied to the most recent shipment of a given type, does not necessarily mean that an older billing code is no longer valid. Likewise, an old billing code that was used numerous times is not necessarily more significant than a billing code used only on the most recent shipments. Nonetheless, frequency and recency do remain as two of the most important measurements when assigning billing codes when conflicting codes are available. An expert in the shipping process is often required to select the correct billing code when all these problems are in play.

Existing techniques for CBR maintenance rely on being able to clearly distinguish between correct and incorrect solutions. Since most of the billing codes in the database were accurate when they were entered, traditional maintenance is not

applicable to this domain. Cycles and other changes to billing codes do not follow a set pattern, so a consistent selection method could not be defined by domain experts. Attempts at maintaining a set of rules for selecting billing codes that take into account billing cycles was expensive to develop, error-prone, and difficult to maintain.

Chapter 5

Methodology

Case Based Reasoning was chosen to do the initial matching based on discussions with individuals who have experience working with our domain. It was chosen after unsuccessful attempts at using rule based systems, which required frequent maintenance and were generally difficult to maintain. By using CBR, we were able to add each case to the case base and keep up to date with changes in the domain. Since we were dealing with an unedited case base on which we wanted to minimize maintenance and maximize how up to date the data was, we knew we would not be dealing with a concise and consistent case base. The size of case base and the repetitive nature of the domain made it clear that sorting through repeated cases and resolving conflicts would be important factors we needed to account for.

The initial CBR matching is a weighted nearest-neighbor algorithm. There are 20 attributes that were considered significant to the matching process. These attributes were assigned weights by domain experts corresponding to their significance. Each of these attributes is symbolic. Each case also contains a timestamp for when the shipment was processed and a billing code. When comparing cases, we consider an attribute to be matching only if there are identical values; there is no partial matching. Additionally, several attributes were considered mandatory for two cases to be considered similar. If values did not match for these attributes, then the cases were

assigned a similarity score of 0. A complete list of case attributes used in the matching process is included in Table 1. An explanation of the attributes is found in Appendix B.

Name	Weight	Match Type
MVMT_REV_CODE	4	MANDATORY_SYMBOLIC
STCC_NUMBER	2	SYMBOLIC
ORIG_333	4	MANDATORY_SYMBOLIC
ORIG_ST	2	SYMBOLIC
DEST_333	1	SYMBOLIC
DEST_ST	1	SYMBOLIC
CUST_633	2	SYMBOLIC
CUST_633_TP	2	SYMBOLIC
CUST_633_CNSG	1	SYMBOLIC
CUST_633_SHPR	5	SYMBOLIC
CUST_633_RU	2	SYMBOLIC
CONTR_ID	6	SYMBOLIC
FGN_CONTR_ID	6	SYMBOLIC
CONTR_QUOTE	2	SYMBOLIC
FGN_CONTR_QUOTE	2	SYMBOLIC
CAR_KIND_AAR	2	MANDATORY_SYMBOLIC
TARIFF_ITEM	1	SYMBOLIC
TARIFF_SUPP	1	SYMBOLIC
TARIFF_AUTH	1	SYMBOLIC
TARIFF_CARR	1	SYMBOLIC

Table 5.1. Case attributes used in the initial nearest-neighbor matching, along with weight and matching type.

To obtain a similarity score for two cases in general, we sum the weights of all the attributes that are used, and divide by the sum of all the weights to obtain a normalized score. Each new case is assigned a similarity score to each case in the case base using this method. Frequently, many cases will be assigned similarity scores above a user-defined threshold by the initial CBR matching. Each case suggests a billing code to be assigned to the current case. When all the cases above the threshold suggest the same billing code, this code can be assigned with a high

degree of accuracy. When conflicting billing codes are returned, however, there is no established method to select the correct code.

When there are many billing codes retrieved, leading to conflicts as to the solution the system should select, our system needs to decide which of the billing codes to pick. To test the feasibility of this process, experiments were performed to determine if a CBR program which uses GA-trained conflict resolution rules can increase the accuracy and coverage of a CBR system on its own. As mentioned above, the CBR system selects the most similar cases based on weighted matching of numerous features. When performing conflict resolution, the selection of the most appropriate case should be based on a combination of the recency of the billing case, its frequency (how often it has been used), and its similarity to the current billing problem. Over a series of meetings with individuals experienced in the domain, a subset of important features and their relative weights was determined. At these meetings, it was also determined that recency and frequency were important features in determining the likelihood of a case being relevant, but no pattern could be easily determined.

Difficulty in expressing relationships like these are common in real world problems. CBR often helps solve this problem because it assesses similarity based on relative importance of features, but it does poorly where the relationship is complex. In our domain, billing patterns that can be both cyclical and shifting. Recency and

frequency were determined to best express these qualities. To include these in the matching process, the GA was developed. At issue was how to combine these parameters in a way that leads to the selection of the correct billing case. We experimented with different equations and weighted parameters using genetic algorithms (GAs). Genetic Algorithms were chosen because they are flexible in the data they can represent, keeping the solution more general, and are good at finding globally near optimal solutions without becoming stuck in locally optimal solutions.

Before we describe the equations we used, we need to define two parameters: *frequency* and *recency*. Frequency is simply the relative frequency of a billing number in the cases retrieved. So, for example, if we retrieve 100 cases that are over the matching threshold, and 60 of them have the same billing number 12345, then the frequency of 12345 is 0.6. To compute the recency of a billing number we first need to define the cut-off distance, in other words the number of days after which a billing number is considered outdated. Each day between the current date and cutoff date is assigned a number between 0 and 1, computed linearly over the time interval. So, for example, if the cutoff date is 5 days in the past, the current date has value 1.0, yesterday has value 0.8, two days ago is 0.6, etc. The recency of a billing number is the normalized sum of the recency value of each case retrieved that has this number.

We defined six different weighted solution selection metrics (WSSMs):

1. A linearly weighted frequency: $\alpha \times frequency - \beta$

2. An exponentially weighted frequency: $\alpha \times e^{-(\beta(1-frequency))} + \gamma$
3. A step function for frequency: $\left\lceil \frac{frequency}{1/\alpha} \right\rceil \times \beta$
4. A linearly weighted recency: $\alpha \times recency - \beta$
5. An exponentially weighted recency: $\alpha \times e^{-(\beta(1-recency))} + \gamma$
6. A step function for recency: $\left\lceil \frac{recency}{1/\alpha} \right\rceil \times \beta$

We also defined three simple retrieval methods to be used as comparisons:

1. Most recent: Select the billing number that is in the most recent case
2. Most frequent: Select the billing number that appears most often in the whole set of retrieved cases
3. K-Nearest Neighbor: Select the billing number that appears most often among the K most similar cases

In this domain, the coefficients that the GA was optimized against are recency and frequency. Based on our knowledge of the domain, they presented the greatest opportunity for improving accuracy. These attributes are domain specific, however, and in other settings, other attributes may provide similar opportunities.

Weighted solution selection metrics show the relative importance of a case's classification compared to that of other cases. WSSMs calculate a score for a

particular solution, not for an individual case. A solution's score is the normalized cumulative for all relevant cases, using one metric. The partial score is found by applying a metric to an individual case, which shows the relative importance of that case in selecting a classification, and multiplying this number by the similarity calculated by the nearest-neighbor matching algorithm. The formula for calculating a partial score for a case *a* is:

$$P = N * F$$

where P is the partial score, N is the score returned by the nearest-neighbor algorithm, and F is the value returned by WSSM being evaluated. The scores for each solution are summed and normalized to determine each classification's score for that WSSM. The formula for calculating the final scores is:

$$S = \left\lceil \frac{\sum P_{c=x}}{\sum P} \right\rceil$$

where S is the final score for a classification, $\sum P_{c=x}$ is the partial score for a case where the classification, *c*, equals *x*, and $\sum P$ is the sum of all partial scores.

The formula of the metric applied is held constant, but the coefficients to the metric are varied by the GA.

After scoring by the WSSMs has been completed, the scores are combined to generate an overall score for the system. Since the scores from the WSSMs have been normalized, we sum the scores using the function: $\sum w * S$ where w is the weight for the WSSM, and normalize the result to get the overall score. Since the

accuracy of the WSSM is not known beforehand, we use a multiplier for each formula to reflect the quality of a given WSSM. Rather than using accuracy of the WSSM during training, we chose to apply another GA to determine the weights. Combinations are a collection of multipliers, one per each formula, that result from applying the GA. This multiplier is applied to the scores for each classification, as generated by a formula. The output for each possible classification is a weighted, combined score from the WSSMs. All the WSSMs are used in a combination, but the weights used, and thus the significance of a WSSM, are varied by the GA.

WSSMs and Combinations output a score for each classification. The program must then determine which classification has the highest score and whether this score is high enough to confidently select the classification. Two selection criteria are used for comparison. The first requires a normalized score of .50, which represents a dominant case. The second criterion is winner takes all, the best score for a classification without a minimum score requirement. The classification accuracy is determined by the application of these criteria to the scores generated by combinations. The same criterion is used in the learning process and experiments.

The minimum similarity threshold over which a case will be added to the pool of potential cases (and its solution to the pool of potential billing numbers) is varied between .90 and 1.00 in steps of .02. In other words, the WSSMs will only be applied to input cases with a similarity score at or above this threshold. An identical learning

procedure is applied at each similarity threshold. The purpose of testing at various thresholds is to find the lowest similarity, and thus greatest number of input cases, for which the program is accurate (since, a lower threshold will allow more cases to be selected).

As mentioned previously, Genetic Algorithms are used to maximize the accuracy of the secondary selection metric by optimizing the coefficients for WSSMs and the weights used for combinations. The learning takes place in two steps; first WSSMs are trained, then, using the resulting coefficients, the combinations are trained. A separate population is maintained for each WSSM at each minimum similarity level. Several test sets are used in the training process, and the GA rotates through them to avoid overfitting. We used a case base of 500 cases for training and another one of 500 cases for testing. The case base was sorted by timestamps, and the first 500 cases were used for training, the last 500 for testing. This was done to simulate how the case base will be built in the real world. The training cases were again divided into five sub-sets of 100 cases each. The GA switched between training sub-sets for each generation, in an effort to avoid overfitting the solution. This process is repeated at each similarity level.

A fitness function is a heuristic used to gauge the overall accuracy of a given genetic strain. In this experiment, the fitness function is always a weighted combination of four statistics gathered by running the genetic strain through the conflict resolution

program. The statistics used are: percent correctly classified, percent incorrectly classified, difference in score between the correct classification and the top scoring incorrect classification when the correct classification is selected, and difference in score between the correct classification and the top incorrect classification when the wrong classification is selected. Two combinations of multipliers are to be applied to these statistics in the training process. The first uses only percentage correctly and percentage incorrectly classified, with incorrectly weighted twice as strongly as correctly. The formula is:

$$\left[\frac{N_c}{N_c + N_I}\right]$$

where N_c is the number of cases correctly classified and N_I is the number of cases incorrectly classified. The second fitness function also uses the difference between score when correctly and incorrectly choosing a classification; the goal was to have high accuracy but also to punish incorrect solutions that matched highly. The formula for the second fitness formula is:

$$\frac{N_c}{N_c + N_I} + D_C + D_I$$

where D_C is the average difference between the score of the correct classification and the top scoring incorrect classification when the correct classification was selected and D_I is the same measure, only when the incorrect classification was selected.

The Genetic Algorithm encodes the parameters which we are trying to learn into a bit string. For the exponentially weighted frequency equation, the GA tried to find the parameters α, β, and γ; for the linearly weighted recency equation, the GA tried to find the parameters α, β, and the cutoff date distance; etc. The numbers generated were floating point with resolution of 0.001. For the exponential formulas, we used 19 bits to encode the α parameter, 17 bits to encode the β parameter, and 20 bits to encode the γ parameter. For the linear formulas, we used 17 bits to encode the α parameter and 10 bits to encode the β parameter. There are also integer parameters. For example, the date length is encoded in 9 bits (the date length is the number of days past when recency is set to zero), and the step number, α, and step height, β, are each encoded in 10 bits. The weight for each formula when combining scores was encoded in a 10 bit string.

Parameter	**Length of bit string**	**Encoding**
Combination Weight	10	Float
Step number α	10	Integer
Step height β	10	Integer
Linear value α	17	Float
Linear value β	10	Float
Exponential value α	19	Float
Exponential value β	17	Float
Exponential value γ	20	Float
Cutoff date	9	Integer

Table 5.2. Parameter encoding properties for the GA.

The parameters for each equation are stored in a single chromosome that is of 90 bit length, with some bits unused. For example, the GA found the following parameter

values for the exponentially weighted recency equation when the similarity threshold was set to 0.90:

cutoff date = 17 days

$\alpha = 0.089$

$\beta = 0.053$

$\gamma = 0.085$

leading to the equation: $0.089 \times e^{-(0.053(1-recency))} + 0.085$

A complete list of the final equations can be found in Appendix C.

At each stage in the training process, the parameters are decoded and a training set is run through the WSSM or Combination being optimized. A fitness function is then applied to the results, to determine the fitness of a strain continuing to the next generation. To populate the subsequent generation, a roulette style selection process is used to determine two mates, with fitness functions, normalized across the generation, being the probability that a given strain for any mating. At each mating, the likelihood for crossover is 99 percent, and the likelihood for mutation is 1 percent. The genetic algorithms are run for 1000 generations per test run. Each generation consists of 200 individuals.

Chapter 6

Experiments

The CBR/GA system was tested using a case set acquired from BNSF. The case set consists of four snapshots of data from between September and November 2001. The snapshots were taken in April of 2002. There are 18591 cases in the case set. The cases in the case set are only instances where an issue occurred in the billing process and corrections needed to be made or the billing code needed to be reassigned. Since we did not have access to the entire database, an assumption was made that problems with shipments that share a billing code tended to reoccur, especially within a short time frame. This fact was confirmed by BNSF shipping experts.

Since the snapshots were taken months after the initial billing, most of the cases had been resolved. Records were included of changes that were made along the way. Of the total cases, 4074 had not yet been resolved, and were eliminated, since we could not confirm correct classification. The average case consisted of 21 fields that were selected from 700 initial fields per billing statement, because they were considered potentially useful to the selection process. Cases that had been corrected contained additional fields for the tracking of changes. A complete list of fields and descriptions is included in the Appendix B.

The cases were sorted by shipment date, and the 1000 most recent cases were removed for training and testing. The first 500 of the selected cases were used for training. These cases were divided into 5 sets of 100 cases each, which were cycled through to prevent overfitting. The second 500 cases were used for testing the accuracy of the system

Each training set is run through the initial CBR matching system and only those cases that have competing classifications are used for actual training. These cases typically amount to about a half of all cases at the .90 similarity, but much fewer were retrieved at higher similarities. The weights assigned to each field when performing CBR matching can be found in Chapter 5, and an explanation of each field is included in Appendix B . All of the fields used in the initial matching are symbols, and matches on fields are boolean. Three fields were identified by BNSF as being mandatory for two cases to match, and these fields are used to automatically eliminate cases from consideration.

After the initial matching process, a set of all the cases from the case base that matched the input case at above a minimum threshold normalized score are associated with the input case. The minimum score is varied from .90 to 1.00 in steps of .02. The only data retained at this point is the timestamp from the case, the initial matching score, and whether it was the correct classification for the training case. All

of the training sets, along with a set of matching cases for each training case, are input to the GA system for training.

The GA training is done in two parts: first the weighted solution selection metrics (WSSMs) are optimized and then the weights for combining the WSSMs are optimized. The chromosomes for the GA are initially set to random values. There are 200 individuals in each GA population. A separate population is maintained for each of the 6 WSSMs at each of the 6 minimum scores. Each generation of training is done using one of the 5 training sets. The training sets are rotated through in a fixed order. In total, 1000 generations are done for a full training period.

Each chromosome in a population represents a set of values for a WSSM. These values are used to generate a score for each shipping code in the set of similar cases input along with the training case to the GA. The shipping code with the highest score is selected. This shipping code is compared to the correct shipping code for the training case, and whichever fitness function that is being tested is applied to determine the health of a given chromosome. The value generated by the fitness function is then applied to produce the next generation of chromosomes. This process is repeated for each case from the test set with each chromosome. Each chromosome for each of the WSSM at each minimum score is processed this way for each generation. After training is completed on the WSSMs, the chromosome with the

highest fitness function from the final generation for each population is selected as the values used for both finding the weights for combining WSSMs and for testing.

After values have been selected for the WSSMs, the process is repeated with the same fitness function to get values for combining the WSSM scores into a final score. Each training case is processed by each WSSM to get a score for each possible shipping code. Each chromosome in a combination population represents a normalized weight for a given WSSM. Each score for a shipping code that was generated by a particular WSSM is multiplied by the combination weight for that WSSM. The weighted scores for a shipping code from each WSSM are totaled. The shipping code with the highest score is selected. This process is repeated for each case in the training set. This is repeated for each chromosome in the population. Then, the fitness function is applied and the next generation is created. This is done for each minimum score. The GA then cycles to the next training set, and the process is repeated for each generation. When training is completed, the weights from the chromosome with the highest fitness function score are selected as combination weights for testing

The training process flows much like the testing process. First, a case is input and scores are generated by the CBR for each case in the case base. Any cases above the minimum similarity are input into the WSSMs to generate a score for each shipping code. These scores are then combined with the weights found in the second half of

the training process. After each shipping code has a final score, the one with the highest score is selected. The shipping code selected is compared to the actual shipping code of the test case to determine the accuracy of the combined CBR/GA system.

The entire training and testing process was then repeated for the second fitness function to compare results.

Chapter 7

Results

The GA program was run for each of the three formulas that used recency and the three that used frequency at each threshold similarity level. The results from these formulas along with those from three more traditional CBR selection methods (K-nearest neighbors, most recent, and most frequent) were then used to run the GA program on the formula that combined the nine other results. This process was repeated with two fitness functions for the GA.

First we will compare the accuracy of the two fitness functions, the results of which are presented in Figure 1. Fitness Formula 1 uses only correct and incorrect classification to assess the fitness of an individual in the GA. We notice that the formulas produced by this Fitness Formula, while accurate at a high similarity, quickly drop in accuracy when presented with cases of lower similarity. Fitness Formula 2, which uses how well the formula differentiates between correct and incorrect cases in terms of score as well as whether the classification was correct or incorrect, does not show such a drop off. The combination formula for Fitness Formula 2 performs similarly well at all threshold values.

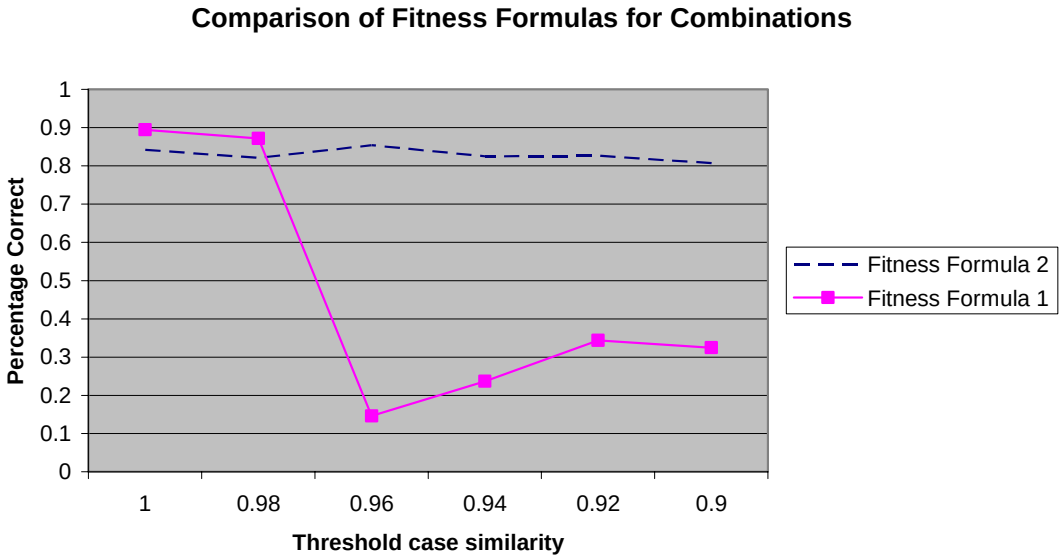

Fig. 7.1. Comparison of the percentage of cases, which had conflicting results at a given threshold similarity, that were correctly classified by the combination formulas for each of the two fitness formulas.

Figure 1 is somewhat misleading in that it does not represent the number of cases that are actually correctly classified. Based solely on the results shown in Figure 1, one may assume that the best performance is to be had by using a threshold case similarity of 1.00 and Fitness Formula 1. Figure 2 shows the percentage of the entire test set that is accurately classified at each threshold value, whether or not there are competing classifications at that threshold for similarity, and compares it to percentage of with competing classifications, which shows the theoretical limit for the classification rate. Looking at Figure 2, we see that Fitness Formula 1 correctly classifies only 5-7% of the test set at threshold similarities 1.00, 0.98, and 0.96, and it classifies at most 19% of test set correctly at threshold similarity 0.90. Fitness Formula 2 classifies a similar percentage at the higher thresholds, but correctly classifies 48% percent of the test set at a threshold similarity of 0.90. The “Cases

with competing classifications" line is included as the maximum percentage correct at each threshold similarity value.

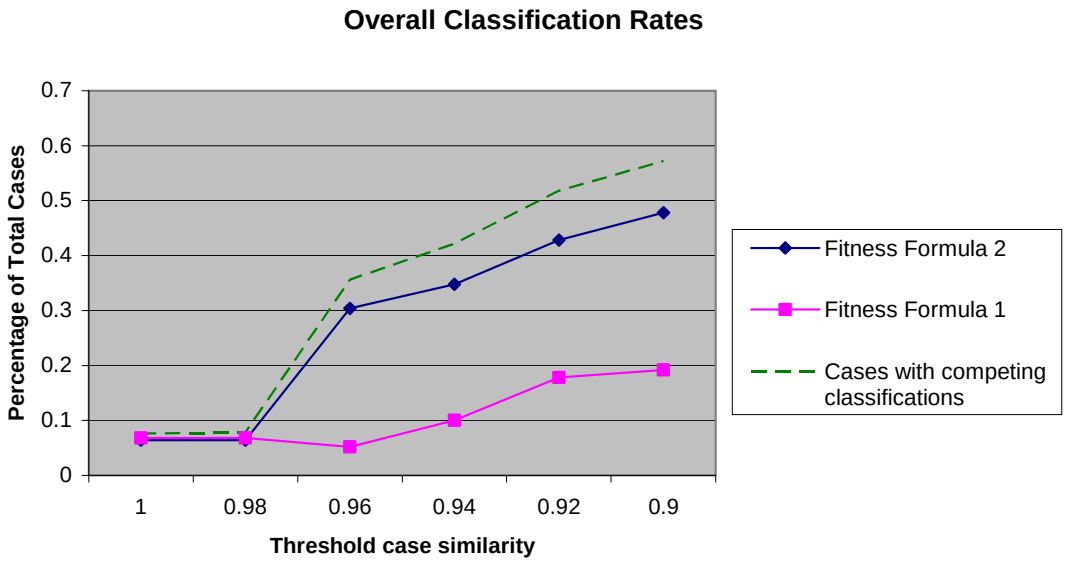

Fig. 7.2. Comparison of the percentage of the test set correctly classified at each threshold similarity by the combination formulas to the percentage of the test set with competing classifications.

Next we will compare the results from base formulas, and how they affected the final results. The first point to be made with regard to Figure 3 is that most of formulas classify cases similarly well. The formulas for "Exponential Recency", "Exponential Frequency", "Most Frequent", and "Step Recency" all correctly classify nearly the same number of cases and "Linear Frequency" follows the same trend, only a few cases lower. This leads us to believe that there is a subset of cases that are relatively easy to classify in spite of having conflicting classifications.

Looking at Figure 3, we also see how strongly related the accuracy of the "Combination" formula is to the "Most Recent" formula. One possible explanation

for this is that "Most Recent" is purely binary in its classification (the score for a given code is either 0 or 1) and this tends to give it too much strength in determining the overall score. If we look at Figures 4 and 5, however, we notice a different cause for the poor performance of Fitness Formula 1. In both cases, we see that the Combination formula has a smaller difference in scores between the correct case and the top scoring incorrect case than do the basic formulas. Fitness Formula 1 takes into account only whether a case was correctly or incorrectly classified in assessing the fitness of an individual in the GA. As a result, there is no advantage for the individual to strongly score a case. When we look at the same statistics with Fitness Formula 2, we will notice that the results for the basic formulas are similar to those of Fitness Formula 1 (see Figures 8 and 9). The difference in scoring for the Combination formula, however, shows that Fitness Formula 2 still strongly differentiates between cases, both when correct and incorrect.

Figures 4 and 5 show the differences between basic formulas in a clearer form. First, we note that, in general, there tends to be a greater difference in scores when the correct classification is made than when an incorrect one is made. This makes sense, since there seems to be a subset of cases that are easily classified, and another set that are more difficult. Also, cases that tend to differentiate between cases strongly when they correctly classify a case tend to do the same when they are incorrect. That is, some formulas seem to be skewed towards giving a smaller group of cases a higher score, while others tend to spread scores more evenly over all competing cases.

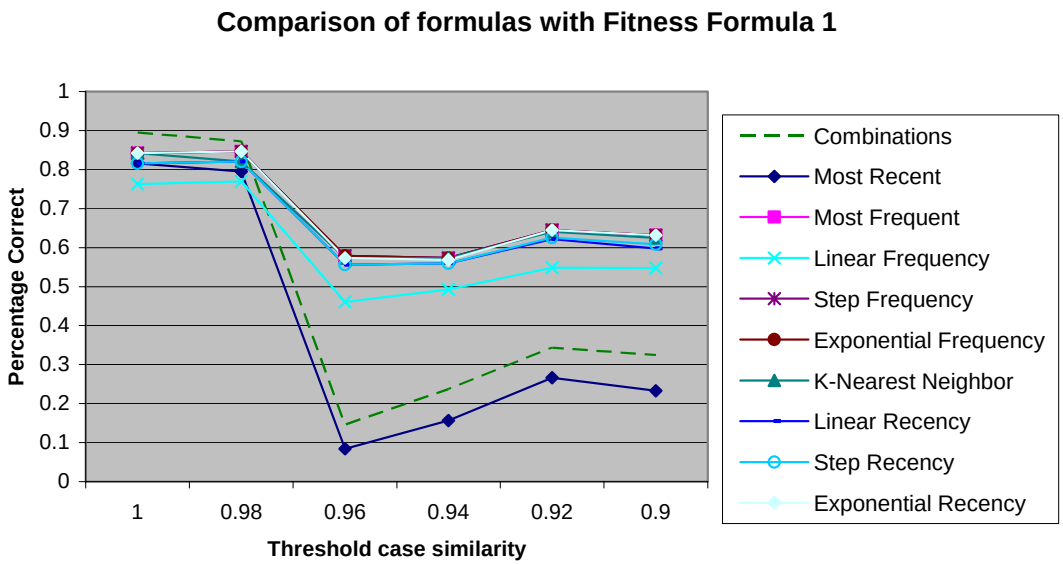

Fig. 7.3. Comparison of the percentage of cases correctly classified by individual formulas trained with Fitness Formula 1.

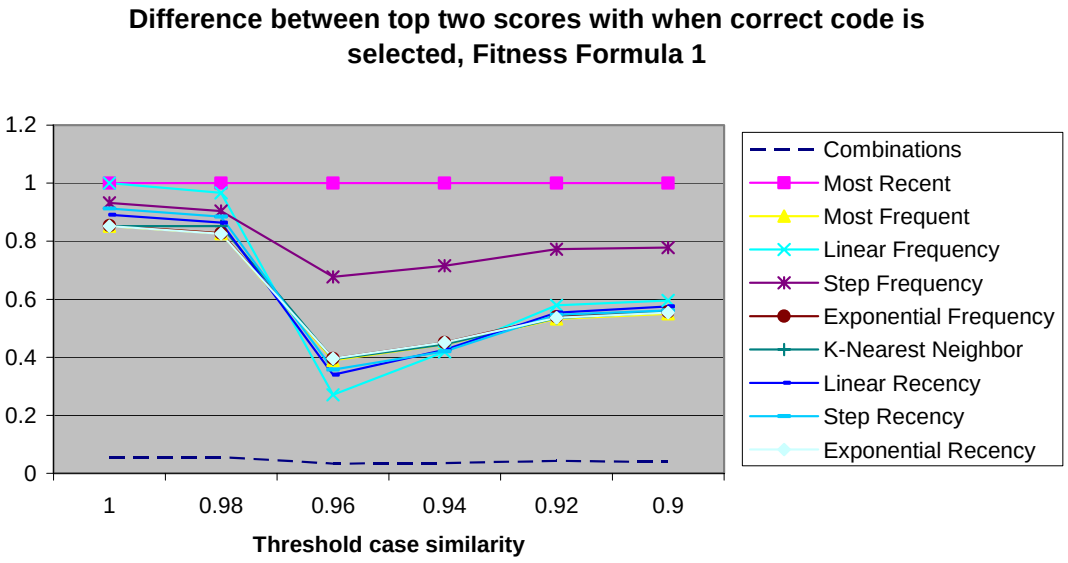

Fig. 7.4. Difference between the score for the correct code and that of the top scoring incorrect code, when the correct code was selected by the formula.

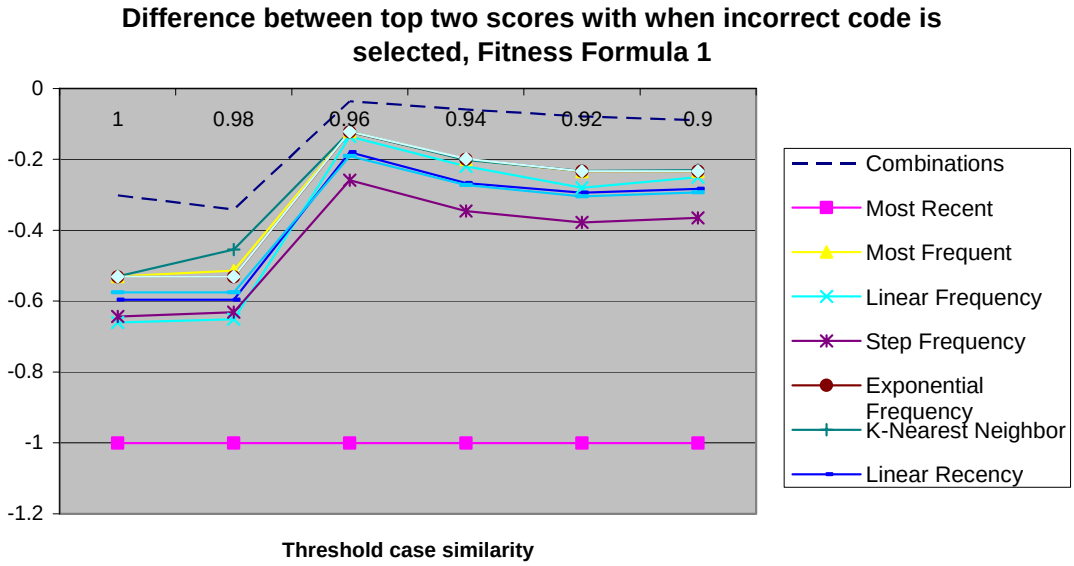

Fig. 7.5. Difference between the score for the correct code and that of the top scoring incorrect code, when the incorrect code was selected by the formula.

If we look at Figure 6, we see the percentage of the test set as a whole that is correctly classified by the individual formulas after training with Fitness Formula 1. This clearly shows how Combination formula shows little improvement in the total number of cases correctly classified as the threshold similarity is reduced. In fact, it lags behind all the other formulas except "Most Recent". It appears that training based solely on correct and incorrect classification is not sufficient.

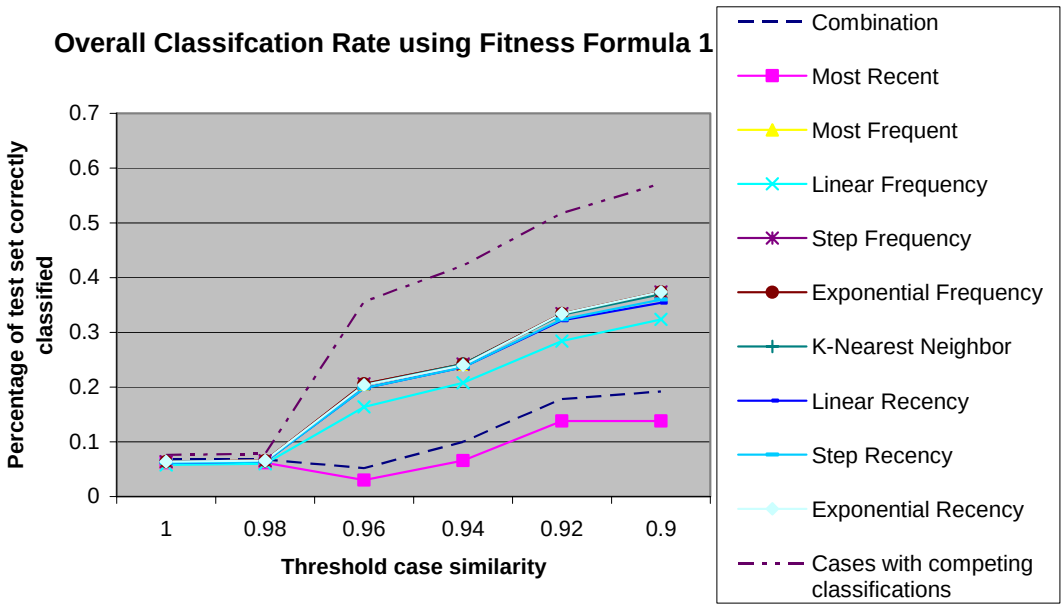

Fig. 7.6. Percentage of cases from the test set correctly classified for each formula at each threshold similarity, using Fitness Formula 1 for training.

Overall, the Combination formula trained using Fitness Formula 1 fails to improve on the standard CBR case selection metrics of Most Frequent and K-Nearest Neighbor. Several of the basic formulas that were trained with Fitness Formula 1 showed similar accuracy to these metrics, but none was able to improve upon them.

Looking at Figure 7, which shows the percentage of cases correctly classified after training with Fitness Formula 2, we see very different behavior. The first important point is that the Combinations formula outperforms all of the basic formulas at threshold similarities 0.96 and below. This contrasts with Fitness Formula 1 (see Figure 3), where the Combinations formula underperformed all formulas except for "Most Recent". Also, note that two formulas, "Exponential Recency" and "Linear

Recency", perform much worse with Fitness Formula 2 than they do with Fitness Formula 1.

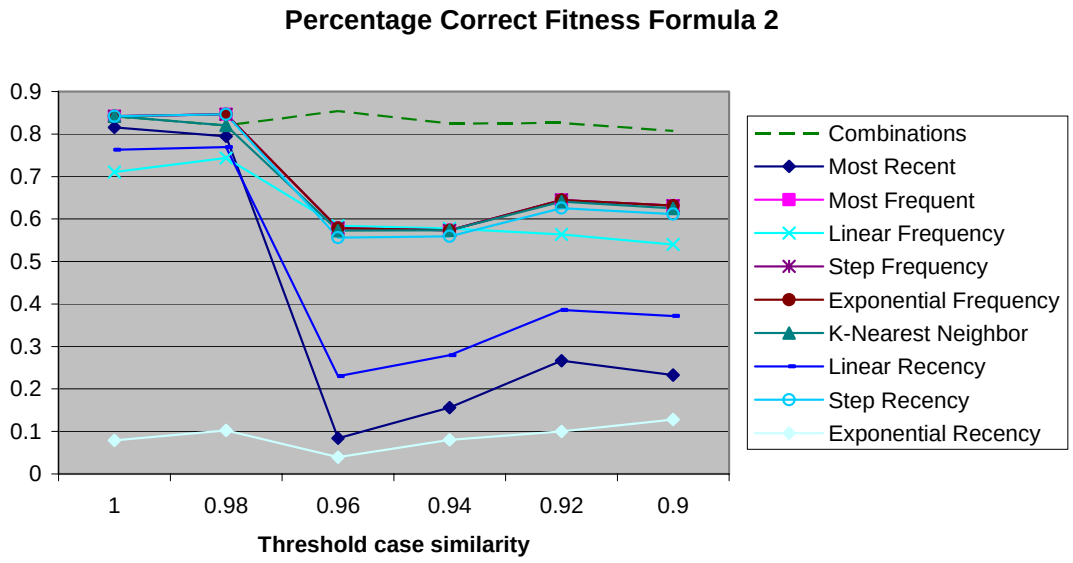

Fig. 7.7. Comparison of the percentage of cases correctly classified by individual formulas trained with Fitness Formula 1.

If we look at the classification accuracy in Figure 7, and compare this to how strongly formulas differentiate between cases in Figure 8 and 9, we see some interesting results.

In Figure 7, we see that "Linear Recency" is a poor classifier, accurately classifying cases less than 40% of the time when the threshold similarity is below 0.98. Figure 8, however, shows us that "Linear Recency" is among the best at differentiating between correct and incorrect cases when it does in fact select the correct case. Looking at Figure 9, we see that when "Linear Recency" selects the wrong case, the scores of the correct and incorrect cases are very close. This is an example of when a formula may not be valuable as a selection method on its own, but does help in the

decision making process. This also leads to the possibility that selection accuracy for some formulas could be improved on by requiring a case to have plurality before selection, rather than the majority implemented in this system.

In Figure 7, we see that two of the basic formulas that perform equally well in classification are “Most Frequent” and “Step Frequency”. Both formulas correctly classified 63% of cases at the threshold similarity of 0.90 and 64% at 0.92. In Figure 8, we see that “Step Frequency” has a larger difference in score than does “Most Frequency” when it is correct (0.69 to 0.55, respectively, at threshold 0.90 and 0.66 to 0.55 at threshold 0.92). In Figure 9, we see that when the formulas were incorrect, the difference in the scores was very small, (-.34 to -.31 at 0.90, and -.32 to -.28 at 0.92).

Similarity	**0.9**	**0.92**	**0.94**	**0.96**	**0.98**	**1**
Formula Name						
Most Recent	0.08854	0.138095	0.428571	0.183168	0.882353	0.475728
Most Frequent	0.13021	0.109524	0.053571	0.158416	0.029412	0.029126
Linear Frequency	0.10417	0.090476	0.133929	0.059406	0	0
Step Frequency	0.0625	0.133333	0.026786	0.089109	0.029412	0.058252
Exponential Frequency	0.16667	0.128571	0.089286	0.069307	0	0.097087
K-Nearest Neighbor	0.03646	0.185714	0.053571	0.089109	0	0
Linear Recency	0.04688	0.061905	0.017857	0.123762	0	0.019417
Step Recency	0.11979	0.080952	0.098214	0.054455	0.058824	0.15534
Exponential Frequency	0.24479	0.071429	0.098214	0.173267	0	0.165049

Table 7.1. This table shows the multiplier applied to each basic formula by the Combination formula for Fitness Formula 2.

We can compare the effect of each formula on the Combination formula on both correctly and incorrectly classified cases by multiplying the average score when

correct or incorrect by the Combination formula weights shown in Table 1. The result of this is show in Table 2.

	0.9	**0.92**
Most Frequent when Correct	0.07161	0.060238
Step Frequency when Correct	0.04313	0.088
Most Frequent when Incorrect	-0.04036	-0.03067
Step Frequency when Incorrect	-0.02125	-0.04267

Table 7.2. The effect of "Most Frequent" and "Step Frequency" on the Combination formulas final result both when the correct and incorrect solution is selected.

We can now look at the contribution that each formula made to the Combination score. We define the formula's contribution as follows:

Contribution=(Score when correct x %Correct) – (Score when incorrect x %Incorrect)

The contribution of Most Frequent and Step Frequency to the final Combination formula score are shown in Table 3. If we compare the contribution of each formula at a threshold similarity of 0.90, we see that Most Frequent contributed 0.01087 more towards the final Combination score. At a similarity of 0.92, Step Frequency contributed 0.01345 more.

Both Most Recent and Step Frequency assign scores to cases based on the same raw data (frequency). In this instance, the trained formula, Step Frequency, performs equally well in classification as the standard CBR formula. At some threshold similarities, however, the formula that was trained by the GA provides a stronger contribution towards the Combination formulas classification.

	0.9	0.92
Most Frequent	0.03018	0.027511
Step Frequency	0.0193	0.040959

Table 7.3. Contribution of Most Frequent and Step Frequency formulas to the Combination formula's final scoring.

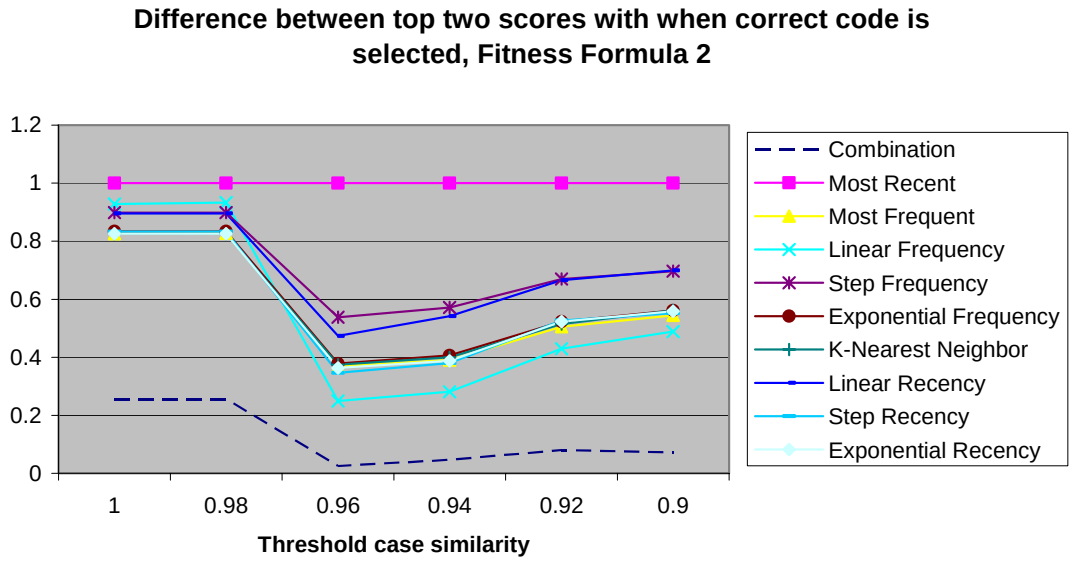

Fig. 7.8. Difference between the score for the correct code and that of the top scoring incorrect code, when the correct code was selected by the formula. (A larger number means that the formula did a better job at differentiating between correct and incorrect codes).

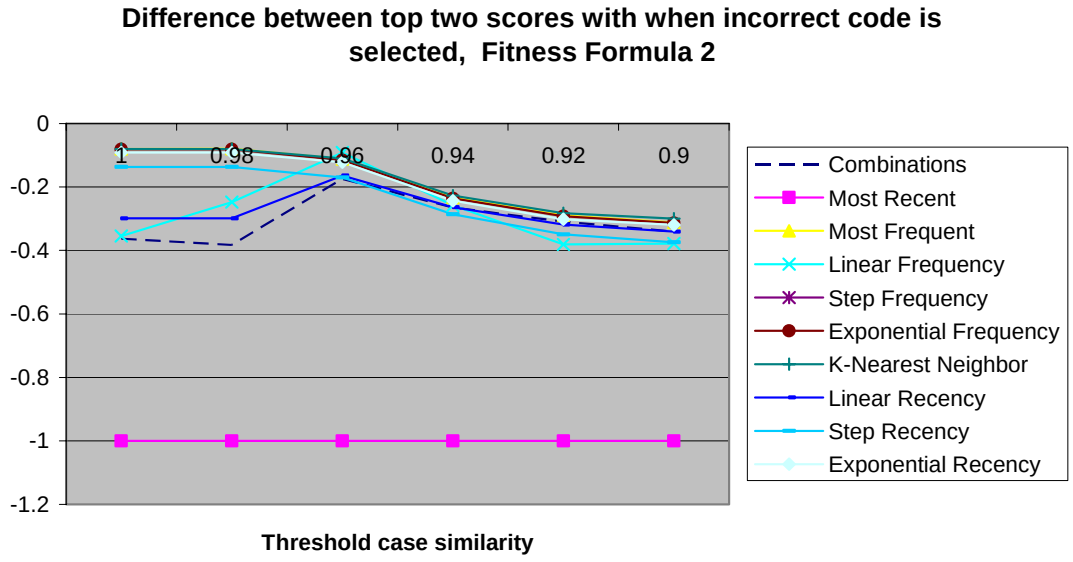

Fig. 7.9. Difference between the score for the correct code and that of the top scoring incorrect code, when the incorrect code was selected by the formula. (Here, values closer to 0 are better, indicating that the correct code scored nearly as well as the incorrect one).

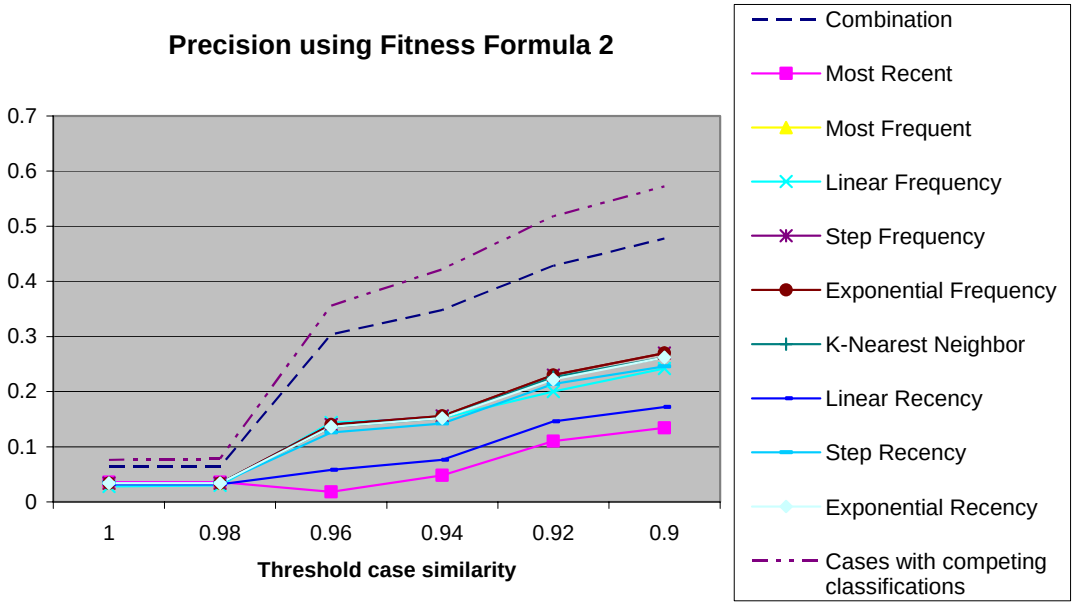

Fig. 7.10. Percentage of cases from the test set correctly classified for each formula at each threshold similarity, using Fitness Formula 1 for training.

We can now look at the performance of the system that combines both the CBR matching, when there are no conflicting solutions returned, and the GA trained portion of the system, as seen in figure 11. The CBR system on its own is able to correctly classify 51% of the test cases at similarity 1.0, but this number declines to 14% at similarity .90.

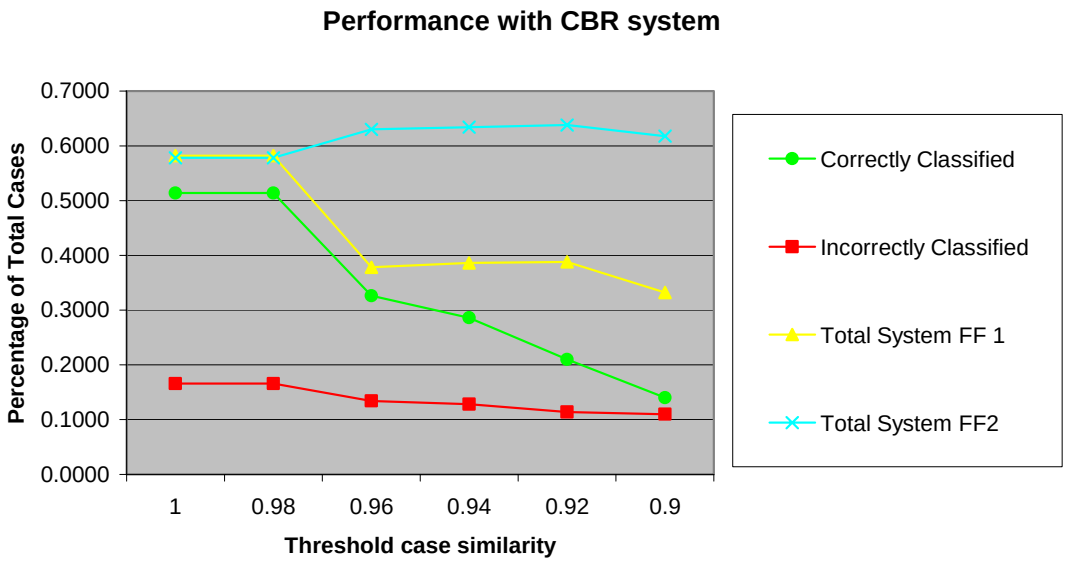

Fig. 7.11. Performance of the CBR system when combining both the initial CBR matching, when no conflicting cases are returned, and the combined formulas learned by the GAs.

We will focus on the performance of the combined system using Fitness Formula 2, since this has been shown to outperform Fitness Formula 1. The combined system performs consistently across all similarity thresholds, correctly classifying between 58% and 63% of the cases from the test set. The combined system performs optimally on the test set using Fitness Formula 2 with .94 as the minimum similarity

threshold. Comparing this to the performance of the CBR system alone, we see that the combined system was able to correctly classify 12% more cases from the test set.

Fitness Formula 1	**1.00**	**0.98**	**0.96**	**0.94**	**0.92**	**0.90**
Classified by CBR	257	257	163	143	105	70
Classified by GA	34	34	26	50	89	96
Incorrectly classified by CBR	83	83	67	64	57	55
Incorrectly classified by GA	4	5	152	161	170	200
Unclassified, no similar cases	122	121	92	82	79	79
Total Test Cases	500	500	500	500	500	500

Table 7.4. Number of cases correctly and incorrectly classified by each part of the system after training with Fitness Formula 1.

The distributions of which parts of the system were used in classifying the cases from the test set are shown in Tables 7.4 and 7.5. Using Fitness Formula 1, the optimal configuration to maximize the number of cases correctly classified is a similarity threshold of 1.00, at which point 281 cases out of 500 were classified correctly, 87 were incorrect and 122 where unclassified because no similar cases wee found. The configuration that performs best in terms of number of correct classification when using Fitness Formula 2 is a similarity threshold of 0.94, at which point 317 cases were correctly classified, 101 were incorrectly classified, and 82 where unclassified. A similarity threshold of 0.96 performs similarly, with 315 cases correctly classified, but reduces the number of incorrect classifications to 93, with 92 cases unclassified.

Fitness Formula 2	**1.00**	**0.98**	**0.96**	**0.94**	**0.92**	**0.90**
Classified by CBR	257	257	163	143	105	70
Classified by GA	32	32	152	174	214	239
Incorrectly classified by CBR	83	83	67	64	57	55
Incorrectly classified by GA	6	7	26	37	45	57
Unclassified, no similar cases	122	121	92	82	79	79
Total Test Cases	500	500	500	500	500	500

Table 7.5. Number of cases correctly and incorrectly classified by each part of the system after training with Fitness Formula 2.

Chapter 8

Conclusion

In some domains where CBR could be useful for problem solving, it cannot be applied because the data set contains insufficient or contradictory data that inhibits proper case selection. This situation is especially common when using real world databases, as the data was not created with this application in mind. Often, selecting between cases is difficult because either the cases do not explicitly contain the information needed to make the decision or because expert knowledge for the domain is difficult to explicitly explain. In this situation, Machine Learning may be applied to otherwise poorly understood concepts to help produce meaningful results.

In our domain, the weakly understood concepts were the use of "frequency" and "recency" to resolve between competing solutions. We were able to elicit enough domain information to perform matching on the database, but there were no explicit rules on how to apply frequency and recency to select between potential cases. By applying GAs to simple formulas, we were able to produce a selection technique that was able to outperform the more standard techniques used for case selection. We found that by combining a variety of formulas to assign a value to these concepts, the system was able to outperform any individual formula, as well as standard selection techniques.

It was possible to get these results because of compartmentalizing the knowledge stored in a CBR system, an applying the knowledge gained during the CBR process to improve the system. It may be possible to apply similar techniques to other domains where poorly understood concepts make it difficult to select between competing cases and where other maintenance techniques cannot be readily applied. In these domains, the concepts may not be frequency and recency. There will likely be other information about the domain that is difficult to explicitly define, but can be of value in improving the case selection process.

Value can be gained by applying our technique to poorly understood corporate domains. Our experiments translate into a potential of 1500 cases being billed correctly per month for BNSF. This results in fewer man hours doing tedious work and more consistent cash flow. The ability to minimize the amount of expert knowledge needed to perform tasks reduces the difficulties in developing CBR classification systems for business applications.

References

Aamodt, Agnar and Enric Plaza. "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches", *AI Communication, Vol. 7 #1,* pp. 39-59, March 1994.

Aha, David. "Case-Based Learning Algorithms", *1991 DARPA Case-Based Reasoning Workshop,* Morgan Kaufmann Publishers, Inc. pp. 147-158, 1991.

Bareiss, E.R., Porter, B.W., Wier, C.C. "Protos: An exemplar-based learning apprentice", *International Journal of Man-Machine Studies 29:* pp. 549-561, 1991.

Cheetham, William. "Case-Based Reasoning with Confidence", *Fifth European Workshop on CBR*. E. Blanzieri and L. Portinale (eds). Springer Verlag: Berlin. pp. 15-25. 2000.

De Jong, Kenneth. "Learning with Genetic Algorithms: An Overview", *Machine Learning 3,* Kluwer Academic: Hingham, Mass. pp. 121-138, 1988.

Globig, Christoph, and Wess, Stefan. "Learning in Case-Based Classification Algorithms", *Algorithm learning for Knowledge Based Systems – Gosler Final Report*, K. Jantke and S. Lange (eds). (LNAI 961). pp. 340-362, 1995.

Heister, F. and W. Wilke. "An Architecture for Maintaining Case-Based Reasoning Systems", *4th European Workshop on CBR*, Springer Verlag: Berlin. pp. 221-232, 1998.

Holland, John H. *Adaptation in Natural and Artificial Systems*, University of Michigan Press: Ann Arbor, Mich, 1975.

Huang, Ye. "An Evolutionary Agent Model of Case-Based Classification", *Third European Workshop on CBR*: Springer Verlag: Berlin, pp. 193-203, 1996

Kelly, J.D. and L. Davis. "A Hybrid Genetic Algorithm for Classification", *Proceedings of the 11th Annual Conference on AI* (IJCAI-91), pp. 645-650, 1991.

Kolodner, Janet. *Case-Based Reasoning*, Morgan Kaufmann Publishers, Inc: San Mateo, CA, 1993.

Kool, K., W. Daelemans, and J. Zavrel. "Genetic Algorithms for Features Relevance Assignment in Memory-Based Language Processing," *Proceedings of the 4th Conference on Computational Natural Language Learning* and *of the 2nd Learning Language in Logic Workshop, Lisbon,* Association for Computational Lingustics: Somerset, NJ, pp. 103-106, 2000.

Leake, David B. "CBR in Context: The Present and the Future", *Case-Based Reasoning: Experiences, Lessons, and Future Directions*: AAAI Press/MIT Press: Menlo Park, pp. 3-30, 1996.

Liu, X. "Combining Genetic Algorithms and Case-based Reasoning for Structure Design," M.S. Computer Science Thesis, University of Nevada at Reno, 1996.

Louis, S.J. and J. Johnson. "Robustness of Case-Initialized Genetic Algorithms", *Proceedings of the 12th International Florida AI Research Society*, FLAIRS-99, pp. 129-133, 1999.

Maher, Mary Lou, and Andres Gomez de Silva Garza. "Design Case Adaptation Using Genetic Algorithms", *Computer Civ. Engineering*: New York, pp. 294-300, 1996.

Oatley, G., J. Tait, J. McIntyre. "A Case-based Reasoning Tool for Vibration Analysis", *Applications and Innovations in Expert Systems VI: Proceedings of the BCS Expert Systems Conference*, R. Milne, A. Macintosh, and M. Bramer (eds.), Springer-Verlag: Berlin, pp. 132-146, 1998.

Perez, E.I., C.A. Coello, and A.H. Aguirre. "Extracting and Re-Using Design Patterns from Genetic Algorithms using Case-Based Reasoning," *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO-2002, Lagdon, W.B. et al. (eds.), Morgan Kaufmann Publishers: San Francisco, CA, pp. 121-141, 2002.

Purvis, Lisa and Salil Athalye. "Towards Improving Case Adaptability with a Genetic Algorithm" *2nd International Conference on CBR* (ICCBR-97), Leake, David B. and Enric Plaza (eds.), Springer Verlag: Berlin, pp. 403-412, 1997.

Racine, Kirsti and Qiang Yang, "On the Consistency Management of Large Case Bases: the Case for Validation", *Proceedings of the AAAI-96 Workshop on Knowledge Base Validation*, American Association for Artificial Intelligence, AAAI-96, pp. 84-90, August 1996.

Ramsey, C.L. and J.J. Grefenstette. "Case-Based Initialization of Genetic Algorithms," *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 84-91, 1993.

Richter, Michael M. "The Knowledge Contained in Similarity Measures", Invited Speech, *First International Conference in Case-Based Reasoning Research and Development*, Sesimbra, Portugal, October 1995.

Roth-Berghover, Thomas and Ioannis Iglezakis, "Six Steps in Case-Based Reasoning: Towards a Maintenance Methodology for Case-Based Reasoning Systems", *Professionelles Wissensmanagement - Erfahrungen und Visionen*, pp. 198-208, 2001.

Schank, R.C. *Dynamic Memory. A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, 1982.

Sengupta, Arijit, David C. Wilson, and David B. Leake. "Constructing and Transforming CBR Implementations: Techniques for Corporate Memory Management", *Proceedings of the 3rd International Conference on CBR*, Workshop II, pp. 9-18, 1999.

Shin, K. and I. Han. "Case-based Reasoning Supported by Genetic Algorithms for Corporate Bond Ratings," *Journal of Expert Systems with Applications 16(2)*, pp. 85-95, 1999.

Skalak, D.B. "Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms," *Proceedings of the 11th International Conference on Machine Learning*, pp. 293-301, 1994.

Smyth, Barry and Padraig Cunningham, "The Utility Problem Analyzed: A Case-Based Reasoning Perspective", *Third European Workshop on CBR*, I. Smith and B. Faltings (eds.), Springer Verlag: Berlin, pp. 392-399, 1996.

Soh, L.K. and C. Tsatsoulis. "Combining Genetic Algorithms and Case-Based Reasoning for Genetic Learning of a Casebase: A Conceptual Framework," *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO-2001, Spector, L. et al (eds.), Morgan Kaufmann Publishers: San Francisco, CA, pp. 376-383, 2001.

Surma, Jerzy and Janusz Tyburcy, "A Study on Competence-Preserving Case Replacing Strategies in Case-Based Reasoning", *Proceedings of the Fourth European Workshop on Case-Based Reasoning*, P. Cunningham, B. Smyth, and M. Keane (eds), Springer Verlag, pp 233-238, 1998.

Watson, Ian. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann Publishers: San Francisco, CA, 1997.

Widmer, Gerhard. "Combining Robustness and Flexibility in Learning Drifting Concepts", *Proceedings of the 11th European Conference on Artificial* Intelligence, ECAI-94, John Wiley and Sons: Amsterdam, pp. 468-472, 1994.

Wilke, W., I. Vollrath, and R. Bermann. "Using Knowledge Containers to Model a Framework for Learning Adaptation Knowledge." *European Conference on Machine Learning (MLNet) Workshop Notes – Case-Based Learning: Beyond Classification of Feature Vectors*, pp. 68-75, 1997.

Wilson, David C. "Case-Base Maintenance: The Husbandry of Experience", Ph.D. Dissertation at Indiana University, 2001.

Zhang, Zhong and Qiang Yang, "Feature Weight Maintenance in Case Bases Using Introspective Learning", *Journal of Intelligent Information Systems*, Kluwer Academic Publishers: The Netherlands., pp. 95-116, 2001.

Appendix A: Example Waybill

PATRON_CODE=999990001
WB_ID=6692-09-02-13.11.55.792001
STCC_NUMBER=20621
MVMT_REV_CODE=PP
ORIG_ST=WA
ORIG_333=HOQUIAM
DEST_ST=TX
DEST_333=BRENHAM
CUST_633=NFIINTLOGIST
CUST_633_TP=
CUST_633_CNSG=
CUST_633_SHPR=
CUST_633_RU=
CONTR_ID=
FGN_CONTR_ID=AGRT5035
CONTR_QUOTE=
FGN_CONTR_QUOTE=QTEHUBU
CAR_KIND_AAR=A
TARIFF_ITEM=1003
TARIFF_SUPP=A
TARIFF_AUTH=80001
TARIFF_CARR=BNSF

Appendix B: Case Attribute Explanations

The following definitions are taken from the BNSF internal SAS System "RAS Data Definitions" document dated Wednesday, February 28, 2001.

CAR_KIND_AAR - AAR CAR KIND CODE TO GROUP CARS BY SIMILAR CHARACTERISTICS. (A000-A999=EQUIPPED BOX CARS, B000-B999=UNEQUIPPED BOX CARS, C000-C999=COVERED HOPPER CARS, D000-D999=LOCOMOTIVE, ETC.)

CONTR_ID - REFRENCE TO CONTRACT OR QUOTE WHICH PUBLISHES THE TRANSPORTATIO N PRICE.

CONTR_QUOTE - APPLICABLE FREIGHT RATE QUOTE NUMBER.

CUST_633 - BNSF STANDARD 633 CODE IDENTIFYING THE CUSTOMER.(GENERATED FROM FULL LEGAL CUSTOMER NAME USING A COMPUTER ALGORITHIM.)

CUST_633_CNSG - 633 SPELLING OF CONSIGNEE CUSTOMER NAME.

CUST_633_RU - 633 SPELLING OF CONSIGNEE CUSTOMER ROUTING NAME.

CUST_633_SHPR - 633 SPELLING OF SHIPPER CUSTOMER NAME.

CUST_633_TP - 633 SPELLING FOR THE PAYOR OF FREIGHT CHARGES.

DEST_333 - 333 SPELLING OF DESTINATION POINT WHERE LINE HAUL SERVICE TERMINATES.

DEST_ST - CODE INDICATING ABBREVIATION OF DESTINATION STATE WHERE LINE HAUL SERVICE TERMINATES. (IA=IOWA, KS=KANSAS, ETC.)

FGN_CONTR_ID - A COMPOSITE IDENTIFIER OF FOREIGN ROAD THE CONTRACT INCLUDING RAMP LOCATION, LEVEL OF SERVICE AND THE CONTRACT NUMBER FOR A FOREIGN ROAD.

FGN_CONTR_QUOTE - APPLICABLE FOREIGN FREIGHT RATE QUOTE NUMBER FOR A FOREIGN ROAD.

MVMT_REV_CODE - CODE INDICATING THE MOVEMENT/SHIPMENT REVENUE CODE. (PP=PREPAID, CC=COLLECT, ETC.)

ORIG_333 - 333 CITY SPELLING OF THE ORIGIN OF THIS SHIPMENT.

ORIG_ST - CODE INDICATING ABBREVIATION FOR SHIPMENT STATE WHERE LINE HAUL SERVICES ORIGINATES. (EX: IA=IOWA, KS=KANSAS, ETC.)

PATRON_CODE - IDENTIFIES THE BILL TO PATRON ASSOCIATED WITH A PATRON CODE. (WHEN A FULL RECYCLE BILL IS ISSUED, IT GOES OUT TO THE REPOSITION PATRON MASTER TO FIND THE 'BILL TO PATRON' ASSOCIATED WITH THE PATRON CODE.)

STCC_NUMBER - STANDARD TRANSPORTATION COMMODITY CODE (STCC) NUMBER IDENTIFYING A COMMODITY. (STCC_NUMBER IS THE COMPOSITE OF PROD_CLS AND PROD_CLS_NBR)

TARIFF_AUTH - IDENTIFIER OF TARIFF AUTHORITY USED FOR FREIGHT CHARGES.

TARIFF_CARR - ISSUING PARTY OR CARRIER REPORTING MARKS

TARIFF_ITEM - TARIFF ITEM NUMBER UPON WHICH A RATE (CHARGE) IS JUSTIFIED OR BASED

TARIFF_SUPP - DOCUMENT ISSUED IN SUPPLEMENT TO A TARIFF - IDENTIFIES SUBSEQUENT ISSUES OF A TARIFF.

WB_ID - SYSTEM TIMESTAMP USED FOR IDENTIFYING A WAYBILL.

Appendix C: Final Formulas

Threshold Cutoff 1.00 – Fitness Formula 1

1. Linearly weighted frequency: $0.064 \times \mathit{frequency} - 0.037$

2. Exponentially weighted frequency: $0.049 \times e^{-(0.075(1-\mathit{frequency}))} + 0.082$

3. Step function for frequency: $\left\lceil \frac{\mathit{frequency}}{1/20} \right\rceil \times 0.017$

4. Linearly weighted recency: $0.048 \times \mathit{recency} - 0.042$ *cutoff date =20*

5. Exponentially weighted recency: $0.086 \times e^{-(0.05(1-\mathit{recency}))} + 0.093$

   cutoff date =31

6. Step function for recency: $\left\lceil \frac{\mathit{recency}}{1/25} \right\rceil \times 0.039$ *cutoff date = 12*

Threshold Cutoff 0.98 – Fitness Formula 1

1. Linearly weighted frequency: $0.09 \times \mathit{frequency} - 0.036$

2. Exponentially weighted frequency: $0.076 \times e^{-(0.071(1-\mathit{frequency}))} + 0.1$

3. Step function for frequency: $\left\lceil \frac{\mathit{frequency}}{1/17} \right\rceil \times 0.032$

4. Linearly weighted recency: $0.078 \times \mathit{recency} - 0.032$ *cutoff date =19*

5. Exponentially weighted recency: $0.1 \times e^{-(0.057(1-\mathit{recency}))} + 0.108$

   cutoff date =29

6. Step function for recency: $\left\lceil \frac{\mathit{recency}}{1/36} \right\rceil \times 0.023$ *cutoff date = 27*

Threshold Cutoff 0.96 – Fitness Formula 1

1. Linearly weighted frequency: $0.02 \times frequency - 0.031$
2. Exponentially weighted frequency: $0.075 \times e^{-(0.06(1-frequency))} + 0.075$
3. Step function for frequency: $\left\lceil \frac{frequency}{1/17} \right\rceil \times 0.026$
4. Linearly weighted recency: $0.019 \times recency - 0.03$ *cutoff date =32*
5. Exponentially weighted recency: $0.11 \times e^{-(0.02(1-recency))} + 0.131$

   cutoff date =5
6. Step function for recency: $\left\lceil \frac{recency}{1/26} \right\rceil \times 0.044$ *cutoff date =32*

Threshold Cutoff 0.94 – Fitness Formula 1

1. Linearly weighted frequency: $0.084 \times frequency - 0.007$
2. Exponentially weighted frequency: $0.107 \times e^{-(0.031(1-frequency))} + 0.148$
3. Step function for frequency: $\left\lceil \frac{frequency}{1/7} \right\rceil \times 0.029$
4. Linearly weighted recency: $0.077 \times recency - 0.025$ *cutoff date =20*
5. Exponentially weighted recency: $0.11 \times e^{-(0.032(1-recency))} + 0.144$

   cutoff date =6
6. Step function for recency: $\left\lceil \frac{recency}{1/28} \right\rceil \times 0.041$ *cutoff date =26*

Threshold Cutoff 0.92 – Fitness Formula 1

1. Linearly weighted frequency: $0.132 \times \textit{frequency} - 0.011$
2. Exponentially weighted frequency: $0.076 \times e^{-(0.078(1-\textit{frequency}))} + 0.064$
3. Step function for frequency: $\left\lceil \frac{\textit{frequency}}{1/44} \right\rceil \times 0.035$
4. Linearly weighted recency: $0.082 \times \textit{recency} - 0.04$ *cutoff date =2*
5. Exponentially weighted recency: $0.123 \times e^{-(0.032(1-\textit{recency}))} + 0.144$

   cutoff date =6
6. Step function for recency: $\left\lceil \frac{\textit{recency}}{1/24} \right\rceil \times 0.037$ *cutoff date =27*

Threshold Cutoff 0.90 – Fitness Formula 1

1. Linearly weighted frequency: $0.017 \times \textit{frequency} - 0.048$
2. Exponentially weighted frequency: $0.134 \times e^{-(0.032(1-\textit{frequency}))} + 0.144$
3. Step function for frequency: $\left\lceil \frac{\textit{frequency}}{17} \right\rceil \times 0.032$
4. Linearly weighted recency: $0.076 \times \textit{recency} - 0.035$ *cutoff date = 18*
5. Exponentially weighted recency: $0.0859 \times e^{-(0.053(1-\textit{recency}))} + 0.085$

   cutoff date = 17
6. Step function for recency: $\left\lceil \frac{\textit{recency}}{1/24} \right\rceil \times 0.044$ *cutoff date = 32*

Threshold Cutoff 1.00 – Fitness Formula 2

1. Linearly weighted frequency: $0.048 \times \textit{frequency} - 0.038$
2. Exponentially weighted frequency: $0.093 \times e^{-(0.138(1-\textit{frequency}))} + 0.0$
3. Step function for frequency: $\left\lceil \frac{\textit{frequency}}{1/5} \right\rceil \times 0.029$
4. Linearly weighted recency: $0.027 \times \textit{recency} - 0.043$ *cutoff date =18*
5. Exponentially weighted recency: $0.098 \times e^{-(0.05(1-\textit{recency}))} + 0.084$

   cutoff date =1
6. Step function for recency: $\left\lceil \frac{\textit{recency}}{1/7} \right\rceil \times 0.02$ *cutoff date =16*

Threshold Cutoff 0.98 – Fitness Formula 2

1. Linearly weighted frequency: $0.039 \times \textit{frequency} - 0.031$
2. Exponentially weighted frequency: $0.107 \times e^{-(0.138(1-\textit{frequency}))} + 0.0$
3. Step function for frequency: $\left\lceil \frac{\textit{frequency}}{1/5} \right\rceil \times 0.022$
4. Linearly weighted recency: $0.063 \times \textit{recency} - 0.014$ *cutoff date =1*
5. Exponentially weighted recency: $0.088 \times e^{-(0.086(1-\textit{recency}))} + 0.061$

   cutoff date =1
6. Step function for recency: $\left\lceil \frac{\textit{recency}}{1/7} \right\rceil \times 0.02$ *cutoff date =16*

Threshold Cutoff 0.96 – Fitness Formula 2

1. Linearly weighted frequency: $0.084 \times \textit{frequency} - 0.007$
2. Exponentially weighted frequency: $0.149 \times e^{-(0.122(1-\textit{frequency}))} + 0.063$
3. Step function for frequency: $\left\lceil \frac{\textit{frequency}}{1/7} \right\rceil \times 0.029$
4. Linearly weighted recency: $0.017 \times \textit{recency} - 0.029$ *cutoff date =25*
5. Exponentially weighted recency: $0.044 \times e^{-(0.006(1-\textit{recency}))} + 0.099$

   cutoff date =31
6. Step function for recency: $\left\lceil \frac{\textit{recency}}{1/1} \right\rceil \times 0.022$ *cutoff date =38*

Threshold Cutoff 0.94 – Fitness Formula 2

1. Linearly weighted frequency: $0.096 \times \textit{frequency} - 0.008$
2. Exponentially weighted frequency: $0.125 \times e^{-(0.124(1-\textit{frequency}))} + 0.064$
3. Step function for frequency: $\left\lceil \frac{\textit{frequency}}{1/7} \right\rceil \times 0.029$
4. Linearly weighted recency: $0.013 \times \textit{recency} - 0.023$ *cutoff date =23*
5. Exponentially weighted recency: $0.004 \times e^{-(0.065(1-\textit{recency}))} + 0.106$

   cutoff date =21
6. Step function for recency: $\left\lceil \frac{\textit{recency}}{1/1} \right\rceil \times 0.011$ *cutoff date =22*

Threshold Cutoff 0.92 – Fitness Formula 2

1. Linearly weighted frequency: $0.06 \times \textit{frequency} - 0.005$
2. Exponentially weighted frequency: $0.112 \times e^{-(0.13(1-\textit{frequency}))} + 0.041$
3. Step function for frequency: $\left\lceil \frac{\textit{frequency}}{1/7} \right\rceil \times 0.018$
4. Linearly weighted recency: $0.016 \times \textit{recency} - 0.031$ *cutoff date =21*
5. Exponentially weighted recency: $0.051 \times e^{-(0.014(1-\textit{recency}))} + 0.102$

   cutoff date =31
6. Step function for recency: $\left\lceil \frac{\textit{recency}}{1/9} \right\rceil \times 0.027$ *cutoff date =9*

Threshold Cutoff 0.90 – Fitness Formula 2

1. Linearly weighted frequency: $0.108 \times \textit{frequency} - 0.005$
2. Exponentially weighted frequency: $0.146 \times e^{-(0.122(1-\textit{frequency}))} + 0.058$
3. Step function for frequency: $\left\lceil \frac{\textit{frequency}}{1/7} \right\rceil \times 0.014$
4. Linearly weighted recency: $0.015 \times \textit{recency} - 0.037$ *cutoff date =20*
5. Exponentially weighted recency: $0.005 \times e^{-(0.068(1-\textit{recency}))} + 0.121$

   cutoff date =21
6. Step function for recency: $\left\lceil \frac{\textit{recency}}{1} \right\rceil \times 0.021$ *cutoff date =26*