Learning Subjective Functions with Large Margins

Claude-Nicolas Fiechter FIECHTER@RTNA.DAIMLERCHRYSLER.COM
Seth Rogers ROGERS@RTNA.DAIMLERCHRYSLER.COM
DaimlerChrysler Research and Technology Center, 1510 Page Mill Road, Palo Alto, CA 94304 USA

Abstract

In many optimization and decision problems the objective function can be expressed as a linear combination of competing criteria, the weights of which specify the relative importance of the criteria for the user. We consider the problem of learning such a "subjective" function from preference judgments collected from traces of user interactions. We propose a new algorithm for that task based on the theory of Support Vector Machines. One advantage of the algorithm is that prior knowledge about the domain can easily be included to constrain the solution. We demonstrate the algorithm in a route recommendation system that adapts to the driver's route preferences. We present experimental results on real users that show that the algorithm performs well in practice.

1. Introduction

In many optimization and decision problems the desirability of a particular solution depends on a number of competing factors or criteria. The solution can be rated along a number of dimensions, and the overall quality of the solution depends on its combined score on all dimension simultaneously.

Most algorithms to solve optimization problems, however, depend on the existence of a single objective function that specifies how good each potential solution is. Multi-criteria problems are typically handled by computing a cost or rating for each criteria independently and combining these costs into a single function.

A particularly simple way of combining the costs is to use a linear combination. In that case the weights in the combination associated with the different costs specify the relative importance of the criteria for the user. A domain expert can sometimes fix these weights, but often the importance of the different criteria is subjective and varies from user to user. We therefore think of the weights as forming a model of the user preferences, and we call such an objective function a *subjective* function.[1]

It is generally difficult or inconvenient to ask the user to explicitly specify the weights for a subjective function. In many cases, the user might not even be consciously aware of the importance he or she gives to the different criteria. One approach to address this problem is to apply machine learning techniques to learn those weights from traces of interactions with the user. In particular, the subjective function can be inferred from user's *preference judgments* that specify, either implicitly or explicitly, that one solution should be ranked higher than another (Cohen et al., 1999). It is often easy to collect these preference judgments unobtrusively, by observing the choices the user makes while interacting with the system.

Gervasio et al. (1999) describe a crisis response scheduling assistant that takes this approach to infer the most appropriate schedule evaluation function for its user. Similarly, in (Rogers et al., 1999) we describe a route advice system that unobtrusively learns a driver's route preferences through interaction with that driver. In this paper we extend that work and describe a new learning algorithm that seems particularly well-suited to learn subjective functions from traces of user interactions.

The learning algorithm relies on the basic ideas of Support Vector Machines (SVM) (Vapnik, 1999; Burges, 1998) but it takes advantage of the particular structure of the problem. It is consequently much simpler than a general SVM. In particular, computing the coefficients of the subjective function only requires solving a linear program, instead of the quadratic programming problem that a general SVM entails. One advantage of the proposed algorithm over other possible machine learning techniques for learning a subjective function is that prior knowledge about admissible coefficients and functions can easily be used to constrain the solution. This is particularly important in user adaptation applications, which often need to infer a user model quickly, from little data.

[1] The term was coined by Justin Boyan.

Below, we first briefly describe our adaptive route advice system, focusing on the route generation component and on the interface that presents the route options to the user and gathers preference feedback. We then formalize the problem of learning the user preference model and describe the learning algorithm. The following section describes some experimental results on real (human) subjects and the conclusion briefly discusses some directions for future research.

2. The Adaptive Route Advisor

The Adaptive Route Advisor is an adaptive user interface (Langley, 1997) that recommends routes on a road network and adapts to a driver's route preferences through interaction with that driver.

2.1 The Routing Algorithm and User Model

The generative component of the Adaptive Route Advisor is a routing algorithm that plans a path through a digital map from a starting point to a destination. The planner represents the digital map as a graph, where the nodes are intersections and the edges are parts of roads between intersections. Our digital maps provide four attributes for each edge: length, estimated driving time, turn angle to connected edges, and road class (e.g., highway, freeway, arterial road, local road). Based on these attributes 14 features of a route are computed, including its estimated driving time, distance, number of left, right, and U-turns, number of intersections, and distance on each road class. The cost of a route is computed as a weighted sum of its features and the system uses an optimized version of Dijkstra's shortest path algorithm (Dijkstra, 1959) to find the path with the minimum cost.

As discussed above, the weights in the objective function play the role of a user model. The system is initialized with a default user model and the model is refined with feedback from interaction with the planner. More specifically, we define an interaction with the planner to be the presentation of a set of N generated routes for a particular routing task and feedback from the user indicating which route is preferable. This is completely unobtrusive to the user, because he or she evaluates a set of routes and selects one as part of the route advice process. From the interaction we derive $N - 1$ preference judgments, representing the fact that the selected route is preferable to each of the presented alternatives, and the learning algorithm processes a sequence of interactions to produce a weight vector that models the preferences expressed. We discuss the learning algorithm itself in Section 3.

In later interactions with that particular user the routing algorithm uses the weights thus computed in its cost function.

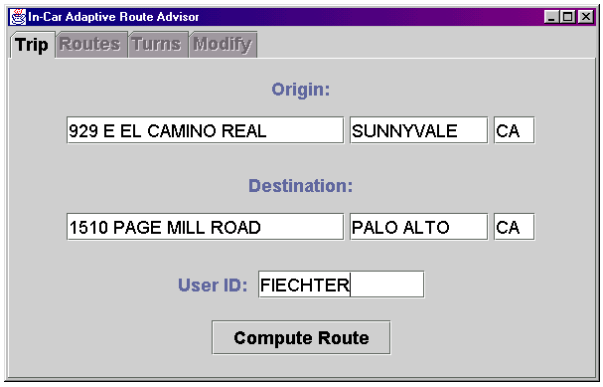


Figure 1. The trip request panel.

In this way, as the driver uses the interface, it adapts itself to his or her preferences. Note that since the routing algorithm is optimal on the cost function, the resulting route is guaranteed to have the lowest cost for that user model among all routes between the same two nodes. In other words, the routes computed are always Pareto optimal, in that there can be routes that are better along each of the dimensions (features) independently, but none that can be better simultaneously on all dimensions.

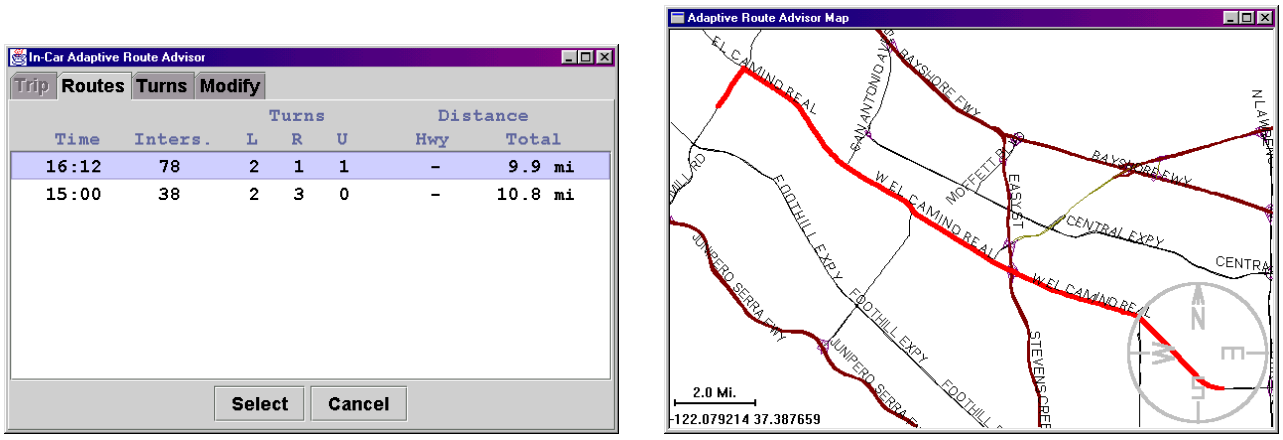
2.2 The Interaction Module

When started, the Route Advisor displays its trip request panel as pictured in Figure 1. In the current implementation, the user specifies origin and destination in a postal address style, and identifies him or herself for the purpose of loading the user model.

After requesting a route, the route summary panel appears, as displayed in Figure 2(a), providing a list of route options. The routes are presented in terms of seven attributes: total time, number of intersections, number of left turns, right turns and U-turns, total distance and distance on highways.

Initially the system computes and presents *two* routes to the user. The first is computed using the current preference model as the weight vector for the routing cost function. It is therefore the route that the system considers optimal for the user. The second route uses novel weights, selected from a small set of prototypical user models. It is inferior according to the current user model but is presented in an attempt to explore new directions in the space of preference models. Presenting at least two route options forces the user to make a choice and provide some feedback to the system.

The map displays the selected route, as shown in Figure 2(b), and the turn directions for the route are available in a separate tabbed panel. Clicking "Select" indicates that the highlighted route is satisfactory and returns to the trip



(a) The route selection panel.

(b) The map window.

Figure 2. Initially, the Adaptive Route Advisor presents two two alternative routes to the user. The best route according to the current user-model is highlighted.

panel. The route advisor assumes that the highlighted route is preferable to the alternative routes and updates the user model. Clicking "Cancel" returns to the trip panel but does not update the model.

The "Modify" panel lets the user generate a new route that is faster, shorter, has fewer turns, has fewer intersections, or has less or more highway than the selected route. The implicit assumption is that the driver is willing to accept routes that are somewhat worse on other attributes if he or she can find one that is better on the selected attribute. The Adaptive Route Advisor searches for new routes that satisfy the improvement request by updating the attribute weights in the appropriate direction. This approach to navigating through the space of possible solutions is similar to "tweaking" in Burke et al.'s RENTME system (Burke et al., 1996). In that system, the user can navigate a database of apartments for rent by asking for an apartment that is either cheaper, bigger, nicer, or safer than the one currently displayed.

The interface described above simultaneously and seamlessly fulfills two functions in the Adaptive Route Advisor. First, it lets the users easily find routes they like by giving them choices and letting them interactively modify the routes proposed. Second, it unobtrusively collects the preference judgments that the learning algorithm needs to refine the user model and adapt to a particular driver.

3. Learning Algorithm

As discussed above, the learning algorithm in the Adaptive Route Advisor processes a sequence of interactions with the planner to produce a weight vector that models the user preferences. These preferences are expressed in the form of pairs of routes $\langle R_1, R_2 \rangle$ for which the user ranked route R_1 as preferable to route R_2.

In general we formalize the problem of learning a subjective function from preference judgments as follows. We assume that each potential solution is characterized by a n-dimensional vector of real numbers that specify the cost of the solution along n, not necessarily independent, dimensions. In general, these costs can be simple numeric attributes of the solution or complex functions computed from several features of the solution.

We are given a set D of m preference judgments $\langle x_d^g, x_d^b \rangle$ $(d = 1, \ldots, m)$, with $x_d^g, x_d^b \in \Re^n$, indicating that the user prefers a solution with cost x_d^g over one with cost x_d^b. Our goal is to determine a subjective function $f : \Re^n \mapsto \Re$ that is consistent with those judgments. Specifically, we would like that $f(x_d^b) > f(x_d^g)$ for $d = 1, \ldots, m$.

Here we restrict ourselves to subjective functions that are linear combinations of the individual costs, i.e., $f(x) = w \cdot x$, with $w, x \in \Re^n$.[2] The vector $w = (w_1, w_2, \ldots, w_n)$ forms our user model and specifies the relative importance of the competing criteria for the user. A large positive value for the weight w_i (relative to the other weights) indicates that the user wants to reduce the associated cost as much as possible, whereas a weight close to zero indicates the user does not mind solutions that are costly along that dimension. In general, if a rational user has the choice between two solutions that differ on a single dimension, he

[2] Since we are only concern with the relative cost of solutions there is no need to include an additive term in the subjective function as it would cancel out.

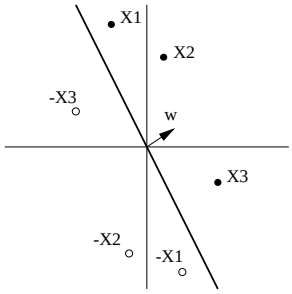


Figure 3. Each preference judgment $\langle x_d^g, x_d^b \rangle$ defines a positive instance x_d and a negative instance $-x_d$ for the linear classifier.

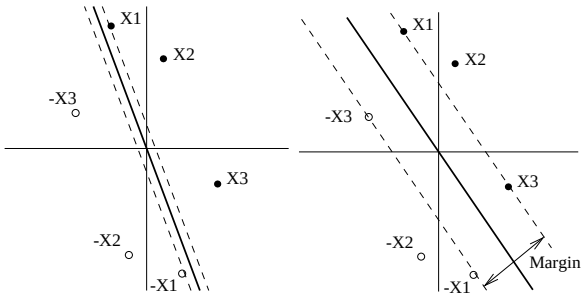


Figure 4. Small margin versus large margin linear classifier.

or she should prefer the solution with the lower cost on that dimension. We therefore assume that the w_i's are non-negative, and do not consider the rare instances where a user truly prefers a higher cost.

The restriction to linear models is not as stringent as it might appear at first, since the cost components themselves can be non-linear functions of the attributes of the solutions. Moreover, we believe that a simple representation, with a limited number of parameters, is generally preferable when learning a user model. Indeed, in those applications, it is often more important to quickly acquire an approximate model of the user preferences from a limited number of user interactions than to achieve the highest possible asymptotical accuracy. The experimental results described in Section 4 illustrate this point.

With this representation, every preference judgment $\langle x_d^g, x_d^b \rangle$ corresponds to a linear constraint on w, namely $w \cdot \left(x_d^b - x_d^g\right) > 0$. We can interpret these constraints as training instances for an induction algorithm that learns a linear classifier w. Specifically, $x_d = \left(x_d^b - x_d^g\right)$ represents a positive instance and $-x_d$ represents a negative instance (see Figure 3).

Any supervised learning technique that is capable of representing a linear classifier can thus be applied to learn the weights of the subjective function from the preference judgments. A previous version of the Adaptive Route Advisor (Rogers et al., 1999), for instance, used a perceptron algorithm.

The algorithm that we describe here is based on the basic ideas of Support Vector Machines (SVMs) and attempts to maximize the margin of the classifier, which is the minimum distance between the hyperplane defined by the classifier and the training points. There is a strong theoretical motivation for maximizing the margin of a classifier that stems from a result in statistical learning theory (Vapnik 1999) that relates a bound on the generalization error of a classifier to the size of its margin. It also has a very intuitive interpretation: classifiers with larger margins are more robust. Both classifiers in Figure 4 correctly classify the training instances, but the second one has a larger margin and is more likely to correctly classify a new instance.

In our setting the margin of a classifier w with respect to a training instance x_d for which $x \cdot w = b$ $(b > 0)$ is given by $2 \cdot b / \|w\|$. Here, $\|w\|$ denotes the Euclidean norm of w. Hence, one way to find a classifier that maximizes the margin is to find a vector w with minimum norm that keeps the separation (b) of all training instances above a fixed threshold.[3] In a general SVM this results in a quadratic programming problem. Here, however, since we know that the components of w must be non-negative we can minimize the 1-norm of w instead of its Euclidean norm to achieve a similar result while solving only a *linear* program.[4] Specifically, we can find a classifier with large margin by solving the following problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} w_j \\ \text{subject to} \quad & w \cdot x_d \geq 1,\ d = 1, \ldots, m \qquad (1) \\ & w_j \geq 0,\ j = 1, \ldots, n \qquad (2) \end{aligned}$$

Note that the above linear program only has a solution when the data are linearly separable. We can easily extend the formulation to the general, non-separable case by transforming the constraints (1) into "soft" constraints and penalizing the constraint violations (Burges, 1998). More precisely, we introduce positive slack variables ξ_d $(d = 1, \ldots, m)$ in the constraints and add a penalty on those variables in the objective function:

$$\text{minimize} \quad \sum_{j=1}^{n} w_j + c \cdot \sum_{d=1}^{m} \xi_d$$

[3] Since jointly scaling w and b does not affect the solution or its margin we can fix the threshold to an arbitrary value and we set $b = 1$.

[4] If $w_j \geq 0$ $(j = 1, \ldots, n)$ then $\sum_{j=1}^{n} w_j / \sqrt{n} \leq \|w\| \leq \sum_{j=1}^{n} w_j$. Therefore, in relatively low-dimensional problems, minimizing $\sum_{j=1}^{n} w_j$ is a reasonable approximation to maximizing the margin.

$$\text{subject to} \quad w \cdot x_d \geq 1 - \xi_d,\ d = 1, \ldots, m$$
$$w_j \geq 0,\ j = 1, \ldots, n$$
$$\xi_d \geq 0,\ d = 1, \ldots, m$$

Here, c is a parameter that specifies the importance of minimizing the training error relative to maximizing the margin.

We can apply a linear programming algorithm, like the Simplex algorithm (see Chvatal, 1983) to efficiently solve the above linear program and compute the coefficients w of the subjective function. One advantage of this approach over other supervised learning algorithms for this problem is that it can be implemented in a truly on-line fashion. Every time the user provides a new preference judgment the system can update the subjective function without having to retrain the learning algorithm on all the data. The system simply adds the constraint associated with the new preference judgment to the linear program and revise the current solution to take new constraint into account. This can be done efficiently within the Simplex algorithm, without having to reconsider all the constraints (Chvatal, 1983). In particular, if the new constraint is satisfied by the current solution w (i.e., if its associated slack variable is null) then w is still optimal and no work is necessary.

The main benefit of the Large Margin algorithm, however, is that prior knowledge about the domain can easily be included in the form of additional constraints on the solution. Such constraints are especially useful when the subjective function must be inferred from a limited amount of data. This will often be the case in the context of Adaptive User Interfaces where we do not want to subject the user to a long "training phase" before starting to adapt to his or her preferences.

In the absence of such constraints, the preferences judgments collected might not be enough to rule out bad solutions. This can be particularly damaging when the inferred subjective function is later used to generate a recommendation. For example, in the Adaptive Route Advisor, we know that regardless of their particular route preferences (e.g., whether they like to drive on highways or not), drivers will always want routes that are reasonably short and fast. However, the previous version of the system would sometime suggest a route that was 50 miles longer because it avoided one mile of local road and previous interactions indicated that the driver disliked those. It is precisely to be able to avoid "aberrant" recommendations like these that the Large Margin algorithm was developed.

In the new version of the Adaptive Route Advisor the type of bad solutions above was prevented by adding to the linear program a constraint that the combined weight of the "total distance" and "estimated time" features in w must represent at least 10% of the total weight. The system includes similar constraints to express other common-sense knowledge about route preferences, like that even a driver that usually prefers route with fewer left turns will not want to do a right turn followed by a U-turn to avoid a left turn.

4. Experimental Results

In order to compare its bias against the route preferences of real users, we tested the Large Margin algorithm as well as several other adaptation algorithms with human participants. Each participant filled out a questionnaire consisting of 20 tasks that involved trips between intersections in the Palo Alto area. We produced four routes for each task using weight vectors with a unit weight for one attribute and zero for the rest. This created routes optimized for time, distance, number of turns, and number of intersections. We plotted the four routes, labeled randomly *A* through *D*, on a map of Palo Alto. To avoid ordering effects, the tasks were randomly reordered for each questionnaire. Figure 5 shows an example of one of the tasks and its four route choices.

We asked the participants to evaluate the routes for each task and rank them in preference order. To control for context (e.g., whether the driver is in a hurry or sightseeing), the participants were instructed to imagine that it is Sunday afternoon, and they are going out on errands. Since an ordering of four items gives six independent binary preferences (A better/worse than B, C, D; B better/worse than C, D; C better/worse than D), each participant provided $6 \cdot 20 = 120$ training instances. The features for each instance were four metrics describing the route: distance, estimated duration, number of turns, and number of intersections. More descriptive features would have been helpful, such as lane merges or traffic lights, but they were not available in our digital map.

4.1 Adaptation Algorithms

Besides the Large Margin algorithm, we tested three other learning methods: perceptron training (Nilsson, 1965) as in the previous version of the Adaptive Route Advisor, SANE (Moriarty, 1997), and a search over lexicographic orderings.

As described in the previous section, both the Large Margin algorithm and the perceptron algorithm implement simple linear classifiers. To test whether a more powerful representation would be beneficial we also implemented a multilayer neural network. The hidden layer in this network combines the input features into possibly more meaningful features, and next combines these features into a route cost. The most natural formulation of the problem, as previously implemented in the perceptron, is to have one input node for each of the route features, and one output node representing the subjective cost of the route. Standard backpropagation, however, is not directly applicable, since only

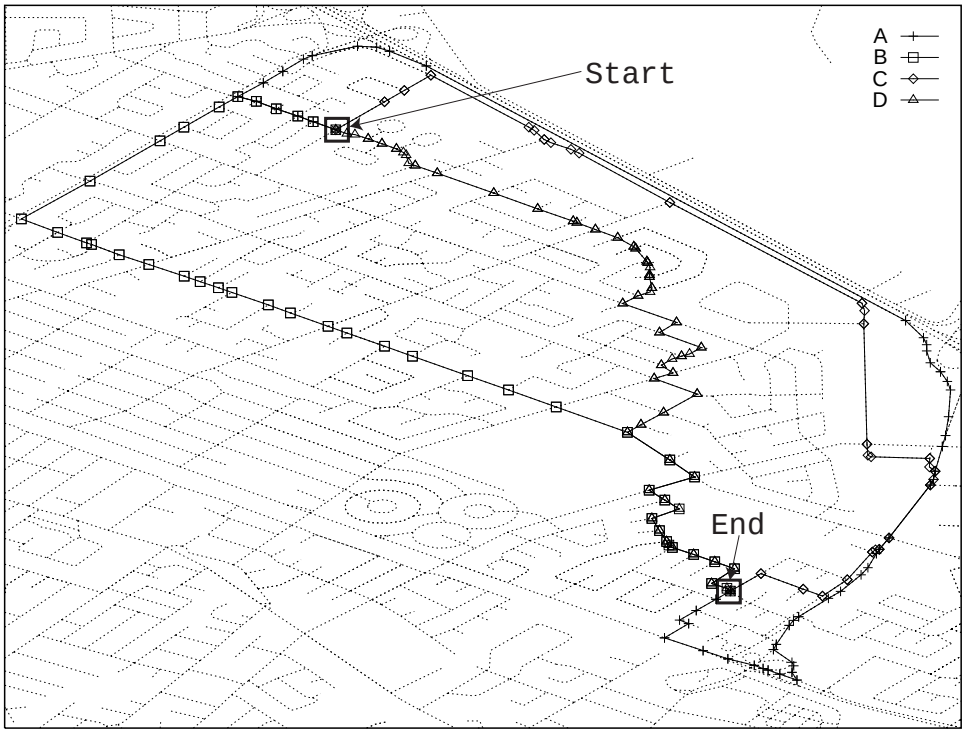


Figure 5. Sample task for the participants. The starting point is the box at the upper left and the ending point is the box at the lower right. *A* is the route with fewest turns, *B* is the fastest route, *C* is the route with fewest intersections, and *D* is the shortest route.

relative rankings, not absolute costs, are readily available from the training data. Instead, we found a more natural training signal to be reinforcement learning-style positive feedback if the relative ordering of two routes is correct or negative if the ordering is incorrect. The SANE (Symbiotic, Adaptive Neuro-Evolution) system (Moriarty, 1997) evolves neural network weights through such weak feedback. SANE is an evolutionary algorithm that manages a population of neurons, constructs neural networks with a fixed configuration from the neurons in each generation, and evaluates the networks. The neurons and weights participating in the most effective networks reproduce to the next generation. SANE's adaptation has been shown to be fast and accurate over several task domains.

Another natural approach to evaluating routes is a lexicographic (or *fuzzy-lexicographic*) comparison on the features. This involves generating an ordering on the features, such as `[Time, Distance, Turns, Intersections]`, and comparing the features of two routes in that order. If a route is (significantly) better than another for one feature, it is preferred. Otherwise, the comparison goes to the next feature in the ordering. In the fuzzy version used here, the amount by which a feature has to be better for the whole route to be considered better is computed as the product of a *significance parameter* λ with the standard deviation σ of the values for that feature. For example, suppose $\lambda = 1$ and time is the most important feature. If there is a route that is less than one standard deviation faster than all other routes, the system predicts the user will prefer it. Otherwise, the system examines the next feature in the ordering. If no single route dominates, the system randomly selects a route. In domains such as this with small feature sets, the adaptation algorithm can find the best ordering by explicitly enumerating all possible orderings.

4.2 Results

Previous experiments with this data set (Rogers et al., 1999) established that personalized cost functions are more accurate than a generic cost function, implying that different users really have different preferences in this domain. The goal of the experiments described here was to find the adaptation algorithm that best induces personalized models for generating satisfactory new routes. Figure 6 presents the testing accuracy of the four adaptation algorithms on each participant. For each algorithm, we measure the accuracy with ten-fold cross validation. The cost c of misclassification in the large margin algorithm was set to 2, and we included all weight constraints described in Section 3. The perceptron was trained for 100,000 epochs with a learning rate $\eta = 0.001$. The lexicographic search estimated the significance parameter λ for each possible ordering via cross validation. SANE trained a network with five hidden units.

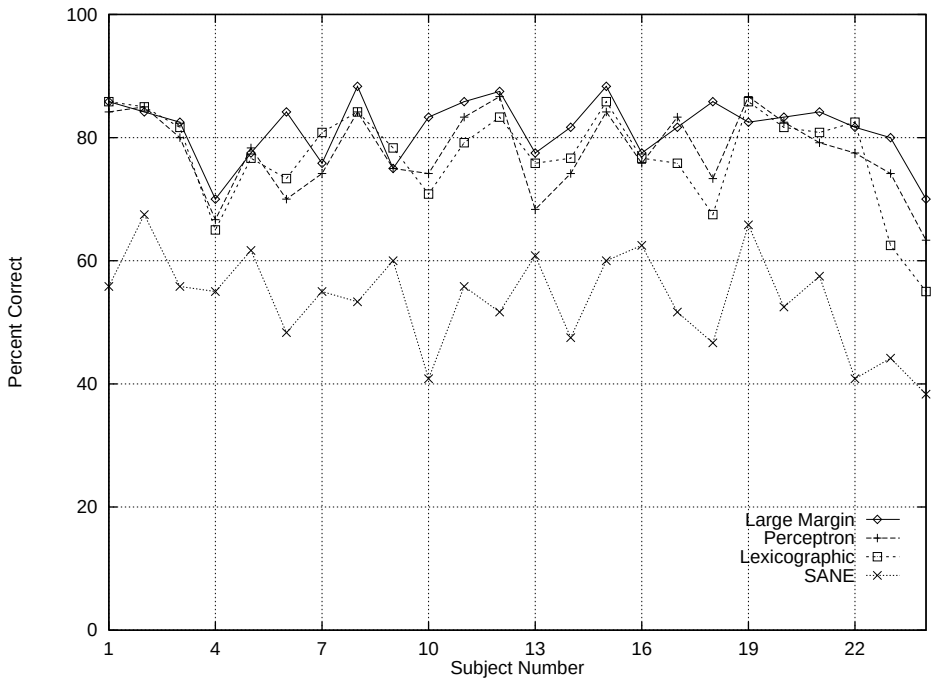


Figure 6. Accuracy of each adaptation technique. The accuracy was computed using ten-fold cross validation.

For this data set, the large margin, perceptron, and lexicographic testing accuracies are very similar. They all performed significantly better than chance (50%), but they did not reach 100% performance, even on the training data. For the large margin and perceptron training, this indicates that the human preference data is not linearly separable.

SANE's accuracy, on the other hand, is significantly lower even though its representation is more flexible. There does not appear to be enough training data for SANE to accurately tune the neural network. This seems to corroborate the intuition that in adaptive user interfaces application, where it is important for the system to quickly acquire an approximate model of the user's preferences, simpler representations can learn more accurate models with little data. Other studies of testing accuracy versus representational complexity for a given training set support this finding, such as the comparison of naive Bayes nets to more powerful classifiers (Langley et al., 1992; Domingos & Pazzani, 1997), as well as the comparison of 1-level decision trees (decision "stumps") to full decision-tree induction (Holte, 1993).

Although the lexicographic ordering performs well on this data set it is not in general a convenient representation to use in a linear optimization algorithm. In the route advisor, for instance, for any given weight vector designed to emulate the lexicographic ordering, there may be a pair of routes that violates that ordering, where the values of a lower-ranked feature of the lexicographically-preferred route is so superior that it overwhelms the first feature. I.e., for any weight vector w and route x, there may exist a route y such that x is lexicographically preferable to y ($x_1 < y_1 - \lambda \cdot \sigma_1$) but $w_1 \cdot x_1 + w_2 \cdot x_2 > w_1 \cdot x_1 + w_2 \cdot y_2$.

By contrast, both the perceptron training and the large margin algorithm use a linear combination of weights to represent preferences, leading to efficient optimal path calculations. Although the accuracies of the two algorithms were comparable, we observed that the model learned by the perceptron for some participants had negative weights on some features. This did not hurt the performance of the perceptron on the testing set, because many cost functions give equivalent results on small testing sets such as ours. However, it means that *higher* values for those features makes the route more desirable, which is quite counter-intuitive and would probably lead to very poor route recommendations on some tasks. Even though the large margin algorithm with constraints did not perform significantly better than perceptron training, we have found that the constraints let the large margin algorithm generate models that are consistent with our common-sense knowledge of route preferences, and therefore more likely to produce good advice.

5. Conclusions

To build systems that help their users solve optimization or decision problems it is often desirable to learn a user's subjective function from preference judgments. We have presented a large margin algorithm that we think is particu-

larly well suited for that task. In particular prior knowledge about the domain can easily be used to constrain the models the algorithm generates.

We have demonstrated the algorithm in the Adaptive Route Advisor system and presented here some experimental results on real users that show that the algorithm is competitive. We have also conducted more systematic experiments on synthetic users that show that the algorithm quickly converges toward good models of the user's route preferences, in a few interactions with the user. These results will be described in a longer version of the paper.

Several researchers have investigated the problem of learning preference models or rankings from ordered pairs of instances. Utgoff and Saxena (1987), for example, use a decision tree induction algorithm to learn a boolean preference predicate $P(x,y)$ that indicates whether a state x should be preferred to a state y in a search control problem. Tesauro (1989) describes a connectionist approach for a similar task, and Cohen et al. (1999) use a version of Freund and Schapire's Hedge algorithm to learn a probabilistic form of a preference predicate, as a step toward identifying a total ordering of the instances. The "State Preference Method" in (Utgoff & Clouse, 1991) is the closest in spirit to ours and tries to identify a linear evaluation function for search control using a form of the perceptron training rule.

The work presented here can be extended in several ways. We are particularly interested in testing the large margin algorithm in other adaptive recommendation systems to see how it performs in different domains. In the Adaptive Route Advisor, we want to address the problem of the context for the route recommendation. The driver is likely to prefer different routes depending on the time of day or whether he or she is sightseeing or going to an important business meeting. One simple approach would be to build a separate model for each context. We are more interested in a solution that would automatically "cluster" the preferences into appropriate contexts.

References

Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*, 121–167.

Burke, R., Hammond, K., & Young, B. (1996). Knowledge-based navigation of complex information spaces. *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (pp. 462–468). Cambridge, MA: AAAI Press/MIT Press.

Chvatal, V. (1983). *Linear programming*. New York: Freedmand and Co.

Cohen, W., Schapire, R., & Singer, Y. (1999). Learning to order things. *Journal of Artificial Intelligence Research*, *10*, 243–270.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269–271.

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, *29*, 103–130.

Gervasio, M., Iba, W., & Langley, P. (1999). Learning user evaluation functions for adaptive scheduling assistance. *Proceeding of the Sixteenth International Conference on Machine Learning*. San Francisco: Morgan Kaufman.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, *11*, 63–90.

Langley, P. (1997). Machine learning for adaptive user interfaces. *Proceedings of the 21st German Annual Conference on Artificial Intelligence* (pp. 53–62). Freiburg, Germany: Springer.

Langley, P., Iba, W., & Thompson, K. (1992). An analysis of bayesian classifiers. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 223–228). Cambride, MA: MIT Press.

Moriarty, D. E. (1997). *Symbiotic evolution of neural networks in sequential decision tasks*. Doctoral dissertation, University of Texas at Austin, Austin, TX.

Nilsson, N. J. (1965). *Learning machines*. New York: McGraw-Hill.

Rogers, S., Fiechter, C.-N., & Langley, P. (1999). An adaptive interactive agent for route advice. *Proceedings of the Third International Conference on Autonomous Agents*. New York: ACM Press.

Tesauro, G. (1989). Connectionist learning of expert preferences by comparison training. *Neural Information Processing Systems* (pp. 99–106). San Mateo: Morgan Kaufman.

Utgoff, P., & Clouse, J. (1991). Two kinds of training information for evaluation function learning. *Proceedings of the Ninth Conference on Artificial Intelligence* (pp. 596–600). Menlo Park, CA: AAAI Press/MIT Press.

Utgoff, P., & Saxena, S. (1987). Learning a preference predicate. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 115–121). Los Altos, CA: Morgan Kaufman.

Vapnik, V. (1999). *Statistical learning theory*. New York: Wiley and Sons, Inc.