Personalization of the Automotive Information Environment

Seth Rogers
Pat Langley
(rogers,langley)@rtna.daimlerbenz.com
Daimler-Benz Research and Technology Center
1510 Page Mill Road, Palo Alto, CA 94304-1135

Bryan Johnson
Annabel Liu
(bhj,aliu)@cs.stanford.edu
Computational Learning Laboratory
CSLI, Ventura Hall, Stanford University
220 Panama Street, Stanford, CA 94305-4115

Abstract

Most research in machine learning applications has focussed on developing a knowledge base from training data, and deploying only the knowledge base in industry. This paper describes a new application of machine learning to *personalization* of complex devices. This task is different from some other learning applications because the learning component itself is deployed in the field, and the application dynamically constructs the knowledge base through interaction with a user. A sample application currently under development is a personalized route advisor for automobile drivers. This system dynamically learns a driver's familiar routes for incorporation into route planning, route description, and destination prediction. We believe that this style of machine learning is an ideal solution to problems where a user needs to configure a device to improve its efficiency but lacks resources to do it manually.

Keywords: personalization, driving assistance, route planning

1 INTRODUCTION

The increasing sophistication of devices currently available to consumers, such as computers and luxury automobiles, gives users the opportunity to tailor their devices to work the way they want. For example, televisions now come with the capability to program interesting channels and cycle through them. However, manually entering channels can be a time-consuming and unpleasant process, especially where the final objective is saving time and effort. In the television example, some users never bother to program their favorite channels because it would involve reading a poorly-written technical manual and possibly calling others for help. Television manufacturers have developed an "autoprogram" function to capture *all* available channels, but cable subscribers find themselves constantly searching through uninteresting options.

Fortunately, in situations such as this, the user can provide feedback to the device in terms of preferences, and this feedback can become training data for a machine learning application, allowing the device to *personalize* its behavior for a particular user. A more intelligent approach to efficiently choosing channels is to monitor the channels people actually watch, and promote all channels watched more than a certain fraction of total watching time to "favorite" status. Conceivably, more sophisticated learning algorithms can use features of the environment such as time of day to dynamically predict the channel probability distribution.

This paper describes a new application of machine learning techniques to automatic personalization of configurable appliances. Section 2 describes some characteristics of the personalization task and some modifications to the traditional machine learning application methodology to accommodate it. Section 3 presents a sample application to learning route knowledge for automobile drivers. A sample planning application in Section 4 is under development to utilize this knowledge. Finally, Section 5 summarizes the major issues and presents some possible future applications for automatic personalization.

2 MACHINE LEARNING FOR PERSONALIZATION

Most applications of machine learning are oriented toward inducing knowledge from objective data. Researchers assume accurate results imply the knowledge is universally valid, and field the knowledge base. For example, Leech [5] applied a decision tree algorithm to predict the quality of fuel pellets for nuclear power plants. Maximizing fuel pellet quality increased the sponsoring company's business by more than ten million dollars per year. These systems typically train on real data offline and the production phase uses only the resulting knowledge base.

A more challenging type of application for machine learning in some ways is learning *subjective* knowledge that reflects a particular point of view. These applications effectively learn the individual preferences of the data source. If the data comes from a single human user, machine learning can automatically acquire the user's preferences and personalize the user's environment. Since these systems are designed to work differently for different viewpoints, it is not practical to gather all possible training data and select the appropriate point of view at production time. Instead, the learning algorithm itself must be deployed and train online as it receives subjective data.

2.1 PREVIOUS WORK

Hermens and Schlimmer report one early application of subjective learning [2] that automatically fills out forms in a database application. This system incrementally builds a decision tree for predicting default field values based on previous experience with the user. The induction algorithm integrates with the database to allow immediate feedback and adaptation. This application reduced keystrokes up to 87% after training.

Other researchers have based personalization applications on the World Wide Web. Mladenic and Mitchell have created a personal Web page recommendation system [7]. This system learns a user's interests by watching the links he or she clicks while browsing, and treats selected links as positive examples and not selected links as negative examples. It uses a binary "bag of words" representation as its feature vector. With the optimal combination of feature size and inductive algorithm, up to 95% of all recommended links are indeed interesting to the user.

Maes uses a slightly different approach in her collaborative filtering work [6]. Instead of silently gathering preference data, the user actively presents ratings of items belonging to some class. Based on these ratings and the ratings of others, the user receives predicted ratings of unrated items. This approach effectively learns to which "interest group" a user belongs with respect to some class, where accuracy depends on the total number of ratings.

These systems highlight some important differences between personalization and objective machine learning applications. The necessity of online adaptation implies that an incremental learning mechanism is preferable, so that the user can take advantage of new results immediately. Also, the term "personalized" implies the existence of a larger system that is configurable to a user's needs, so the knowledge is used in service of some task. Finally, evaluation of such an embedded personalizer is difficult for several reasons. Since each user has individual preferences, the performance of the system as a whole must be measured over multiple users as well as over time as the system trains. The performance measure is the marginal benefit of the system to the user compared to the user doing the same tasks without the system. This metric must be measured in the field on an individual basis, and it may be contaminated by the features of the host application. More examples of relevant past work is in the 1996 AAAI Spring Symposium on Acquisition, Learning & Demonstration: Automating Tasks for Users [1].

2.2 METHODOLOGY

The emphasis on online adaptation requires a different methodology than other machine learning applications [4]. The first two steps are still formulating the problem and determining the representation. The problem can be as simple as classification or as complex as inducing a structured computer program to mimic a person's performance. Both tasks share the need to train on user data, so they both qualify as personalization tasks. The representation depends on the problem, but efficient access and modification is more important in personalization because learning generally occurs online.

Once the problem and representation are set, one or more learning algorithms for solving the problem from data are chosen, and they run on sample training data. After suitable refinement and performance evaluation, the learning system itself is fielded.

Fielding is much more complicated than objective machine learning applications because the entire learning

process must be automated and integrated with the application to be personalized. The system must automatically acquire training data, automatically invoke the induction at the appropriate time, and automatically evaluate and present the results in a useful form. In contrast, standard objective applications may receive hand-coded training data, the inductive process is manually repeated until results are satisfactory, and the final representation may be as simple as an instruction sheet on the factory floor.

Evaluating the performance of the personalization application after fielding is similarly complicated. Scientific evaluation, where possible, consists of comparing some performance measure with the personalization and without, either comparing among different users or for the same user before and after personalization. Although personalization clearly has the potential for enormous improvements in efficiency, it is always necessary to make some assumptions in the testing methodology. For example, if a subject improves efficiency after using the personalization application, we must assume that the personalization caused the efficiency increase.

3 LEARNING ROUTE KNOWLEDGE

Although office and personal computing have received the most attention for personalization applications, personalizing other device controllers has great potential, such as adapting automatic gearbox controllers to individual driver's behavior [3]. Another application is automated in-car driving assistance. For example, such an application with access to a digital map database can help the user plan a route to a destination. If the user (driver) has some knowledge of the area surrounding the current location and the destination, personalization should allow the route planner to bias the route toward familiar roads if the driver wants to minimize his probability of getting lost, or toward unfamiliar roads if the driver wants to explore. Other possible applications of personalization include predicting where a user is going and when he will arrive for several possible routes, and summarizing familiar parts of long route plans. All of these applications require the acquisition and representation of the driver's route knowledge. Our system, RouteCompiler, subdivides the problem into two parts: generating the roads driven from Global Positioning System (GPS) readings, and learning which routes (road segment sequences) the driver uses between particular origins and destinations.

3.1 GENERATING ROADS

For the task of learning what roads are familiar to the driver, the input consists of a list of GPS readings, and the output is a directed graph. RouteCompiler works by taking each of the GPS readings in sequence and determining whether it lies at an intersection, in which case it creates a node in the graph, or between two intersections, in which case the reading is part of an edge in the graph.

RouteCompiler makes several simplifying assumptions about the real world and driver behavior that may cause the resulting graph to lose accuracy. It assumes that the real world consists of roads that are straight lines, and that all intersections are separated by a distance greater than the accuracy of the GPS readings. Also, it assumes that travel by the driver is continuous between stopping points, and that therefore movement only stops at intersections or destinations.

3.1.1 The Road Representation

The graph module take a series of position and time readings as input. The system can deal with discontinuities in the data, both geographically and temporally, although the completeness of the resulting graph will obviously suffer. Thus far it uses the time data only to determine the sequence of measurements, and to detect intersections based on the times when the vehicle is stopped.

The output is a directed graph representing the driver's route knowledge. The nodes of the graph correspond to turns or intersections in the real world. The edges of the graph correspond to paths between adjacent nodes. At present RouteCompiler only stores the physical location for each node, and only the starting and ending nodes for each edge.

3.1.2 Converting from Points to Graph

RouteCompiler works by examining each new reading in order, and then determining whether it fits in the partial graph structure which has already been constructed, or whether a new node or edge must be created. The process breaks down into three general steps, which the algorithm performs in sequence for each reading.

First, RouteCompiler checks the new position and time against the prior one to see if they are identical. If they are, then it simply discards the new reading,

since it contains no new information. If the two readings are not identical, the algorithm checks them for geographic or temporal discontinuity. If such a break exists, then the new reading is either part of the graph or a new node.

If the new point represents neither an identical point nor a discontinuity, then RouteCompiler evaluates the new point against the current graph structure to see if it is in a known node or edge. If so, then if the previous point was also part of the known graph, then nothing needs to be done. If the previous point was part of a new edge, then the algorithm adds the completed new edge to the graph.

If the new point is not part of the known structure, then RouteCompiler is in the process of extending an edge or creating a new edge. It evaluates each new point in terms of its distance from the line fitting the points which make up the edge thus far. If the distance is above a certain threshold, then it creates a new node and a new edge; otherwise we just extend the edge.

After the graph-building algorithm executes, the resulting graph will contain a set of nodes representing points where the GPS data starts, stops, turns, or is discontinuous, and a set of edges, representing the GPS points between two nodes.

3.2 LEARNING ROUTES

Given a directed graph of roads familiar to a driver, it is possible to generate a grammar for possible routes between two points. We represent the route knowledge as a context-free grammar instead of a simple finite state machine to capture knowledge at varying levels of detail.

3.2.1 The Route Representation

The input to this algorithm is a single origin and single destination directed graph. From the graph, we can find for each node in the graph all nodes following that node, which we call descendants.

Given the source, destination and the graph, the algorithm generates a context-free grammar to capture the route knowledge of the paths driven by the driver. In this grammar, the terminal and non-terminal symbols are pairs of nodes. An example of the input and output is in Figure 1.

In Figure 1, p(a j) is a non-terminal symbol that represents the path from a to j. We will use (a j) as a shorthand for entire graph. Likewise, p(b h) is another non-terminal for the sub-path from b to h and p(b g) a shorthand for sub-graph from b to g. p(b c) is a terminal symbol for the arc from b to c.

3.2.2 Extracting a Grammar from a Graph

A rewrite rule describes a path from the first node in the symbol to the second node. RouteCompiler creates rewrite rules to consolidate chains of nodes and merge multiple branches from a node.

Consolidation is a linear-time process that starts from one node and continues toward the destination. When the algorithm reaches a branch-out node, it stops temporarily and calls the merging process to remove the branching. When merging process returns, the consolidation continues. RouteCompiler generates the complete grammar by consolidating the graph between source and destination. In Figure 1, the rewrite rules for p(a j) and p(h j) are the result of consolidation.

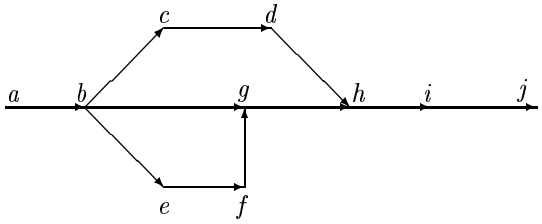
If a node has more than two descendants, RouteCompiler performs pairwise merging. A side effect of the pairwise merging is replacing the two branches by one arc that goes from the node to the converging node of the two branches. This merging process stops when the number of outgoing arcs is reduced to one, i.e. there is no more branching. If there are more than two branches from a node, the pairwise merging is ordered so that the pair of branches that converge earliest merge first. In Figure 1, the rewrite rules for p(b h) and p(b g) are the result of merging.

The final grammar describes possible routes at different levels of abstraction. The most abstract route is any path from the origin to the destination, and the route becomes more specific after each application of a rewrite rule.

3.3 EVALUATION

Like other applications of personalization, performance evaluation is difficult. However, it is possible to evaluate the route knowledge itself. The algorithms trained on both synthetic and real data. A simulated GPS locator generated noisy synthetic data from traversing random maps. The real GPS data source was in a car in eastern Washington state, subject to the standard Selective Availability error characteristics. The data spanned several hours of driving and consisted of hundreds of points.

Comparing the directed graph road representation to a digital map of the area can verify that nodes correspond to intersections and edges correspond to roads.




```
p(a j) :- p(a b) , p(b h) , p(h j) .
p(b h) :- p(b g) , p(g h) .
p(b h) :- p(b c) , p(c d) , p(d h) .
p(b g) :- p(b g) .
p(b g) :- p(b e) , p(e f) , p(f g) .
p(h j) :- p(h i) , p(i j) .
```

Figure 1: A sample graph and its corresponding grammar

Experiments with synthetic maps show that RouteCompiler is very sensitive to noise in the GPS readings. With a mean error of 15 meters, there are approximately 1.5 times as many "false intersections" as real ones, although most true intersections are identified. As accuracy decreases, the error ratio increases linearly. On real GPS data, node errors average approximately 60 meters North-South and 80 meters East-West, which is consistent with an average error of about 100 meters in the GPS data itself. Future work using the digital map to correct GPS errors should improve the accuracy of the graph.

The hierarchical nature of the route grammar is designed to identify important intersections and subdivide routes at these nodes. Therefore, the grammar should include paths between important intersections higher in the parse tree. In a subset of real GPS data where a driver used three routes between two nodes, the mean height for four-connected nodes was 1, three-connected was 1.25, and two-connected was 2.17.

4 APPLICATIONS: FUTURE WORK

The final evaluation must test the knowledge on some performance task. The route planning application will use the route grammar to bias a path toward familiar roads. Since nonterminals in the grammar are assumed to be "chunks" or macros familiar to the user, the nonterminals themselves are added to the digital map as primitive edges. Dykstra's algorithm searches the graph for the least expensive path using a cost function for each edge. Unlike typical path planning edge costs, each edge cost has two components, an estimated travel time and the level of the edge in the grammar, where unfamiliar edges are level zero. A weighting parameter F combines the costs

$$= F + (1 - F)$$

For $F = 1$, familiar roads are always preferred when available. For $F = 0$, the algorithm does not consider familiarity. The weighting parameter also depends on the personal preferences of the driver and can be set manually or learned. This planning technique tends to plan at the most abstract levels of the driver's route knowledge, so it describes the path abstractly as well, and interaction with the driver adds detail on-demand.

The evaluation criterion for this enhanced route planner is the satisfaction of the driver with the proposed route. Since this is not possible to measure directly, one possible indirect measurement is providing drivers with a number of possible routes. For example, the drivers could receive the shortest route, the route with fewest turns, and the most familiar route. Whichever route actually driven is assumed to be the most satisfactory. However, it may be possible to detect that a driver is unsatisfied with his choice if he/she leaves the chosen route at some point or otherwise indicates difficulty with the route. In any case, it is clear that evaluation of performance in this task is more challenging than machine learning applications with a universal standard of truth.

5 CONCLUSION

This paper has identified an important application type for machine learning that has not been as yet widely recognized: personalization. The primary distinguishing feature between personalization applications and objective applications is that the training data for personalization by definition is not available until the application is fielded, since the user him/herself trains the system. This requires developers to be more careful designing the system, because they will not have the opportunity to test the learning algorithm when fielded before the users are affected by the results of the system.

Although more difficult, this machine learning appli-

cation type has potential in many areas. We believe that one of the most interesting of these applications lies in the automotive domain. Just as drivers customize the physical environment in the car's interior by adjusting seats and mirrors, personalization should also let drivers customize their information environment. One important example involves personalization of route planning to incorporate familiar routes. This personalization can be embedded in interactive route planners available today, but future information environments may need personalization in areas like preferred restaurant types, preferred driving style, and preferred maintenance schedule. Over time, personalization will turn a generic factory-tuned information assistant into a pro-active, indispensable partner for mobile efficiency.

References

[1] AAAI. *Acquisition, Learning & Demonstration: Automating Tasks for Users*, Spring Symposium, Menlo Park, CA, March 1996. AAAI Press. Technical Report SS-96-02.

[2] L. A. Hermens and Jeffrey C. Schlimmer. A machine-learning apprentice for the completion of repetitive forms. *IEEE Expert*, 9:28–33, 1994.

[3] K. J. Hunt and R. N. Shorten. Adaptive automatic gearbox control using splines and radial basis function neural networks - concepts. Technical report, DaimlerBenz Berlin, Berlin, March 1997.

[4] Pat Langley and Herb A. Simon. Applications of machine learning and rule induction. *Communications of the ACM*, 38:55–64, November 1995.

[5] W. J. Leech. A rule-based process control method with feedback. *Advances in Instrumentation*, 41:169–175, 1986.

[6] Patti Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, July 1994.

[7] Dunja Mladenic. Personal webwatcher: Implementation and design. Technical Report IJS-DP-7472, Carnegie Mellon University, Pittsburgh, PA, October 1996.