Learning Driver Preferences of POIs Using a Semantic Web Knowledge System

Rahul Parundekar and Kentaro Oguchi

Toyota InfoTechnology Center, U.S.A.
Mountain View, CA
{rparundekar,koguchi}@us.toyota-itc.com

Abstract. In this paper, we present the architecture and implementation of a Semantic Web Knowledge System that is employed to learn driver preferences for Points of Interest (POIs) using a content based approach. Initially, implicit & explicit feedback is collected from drivers about the places that they like. Data about these places is then retrieved from web sources and a POI preference model is built using machine learning algorithms. At a future time, when the driver searches for places that he/she might want to go to, his/her learnt model is used to personalize the result. The data that is used to learn preferences is represented as Linked Data with the help of a POI ontology, and retrieved from multiple POI search services by 'lifting' it into RDF. This structured data is then combined with driver context and fed into a machine learning algorithm that produces a statistical model of the driver's preferences. This data and model is hosted in the cloud and is accessible via intelligent services and an access control mechanism to a client device like an in-vehicle navigation system. We describe the design and implementation of such a system that is currently in-use to study how a driver's preferences can be modeled by the vehicle platform and utilized for POI recommendations.

Keywords: Semantic Web, Linked Data, Knowledge Base, Preference Modeling.

1 Introduction

The in-vehicle navigation system tries to minimize driver distraction by enforcing restrictions on the way information is presented to the driver. One such constraint restricts the number of items that can be displayed in a list on a single screen to a fixed number of slots. When the driver searches for banks in the car, for example, the search results are displayed as a list filled in these slots. If the number of search results exceeds the number of slots, then the extra results are pushed to the next page. A more personalized experience can be provided by showing these results sorted according to the driver's preferences. Using the history of Points of Interests (POIs) that the driver visits, we can build a model of what kind of places he/she (henceforth referred to as *he*) is more likely to prefer. For example, the driver may have certain preferences for banks. We want to understand his bank preferences so that they can be used to personalize the result the next time he is searching for one. Our system, which uses a Semantic Web Knowledge System to explore this personalization, is called Semantic User Preference Engine or *Supe*. The places that the driver visits are stored in our Knowledge Base as data that is structured using RDF. We first use a Machine Learning algorithm to build a preference model of the places that the driver likes from this history. The next time a driver searches for places of a certain category, we use this model to re-order the search results according to his/her estimated affinity.

We describe how Supe is used to study driver preferences of POIs in this paper as follows. First, we discuss our system and how it relies on RDF and Linked Data to represent information. Then, we describe how our the ontology and the Linked Data about the preferred POIs is collected from multiple sources and used to build the preference model for the driver. This includes how the Linked Data is translated into the desired input for Machine Learning algorithms that were used. This is followed by describing how Supe uses the built preference model to reorder a result of a POI search to match the driver's estimated affinity of these places. We then explain the implementation and the evaluation of our prototype system. We also include relevant work in using Linked Data and Semantic Web for modeling user preferences and recommendations. Lastly, we conclude by summarizing our findings along with future work.

2 The Supe System

2.1 Overview

The primary objective of Supe is to collect driver preferences and use the preference model to provide personalized POI search results to the driver. To be successful in modeling driver preferences, it needs to understand the driver as well as the POIs. By defining the semantics in a Knowledge Base, we tie the

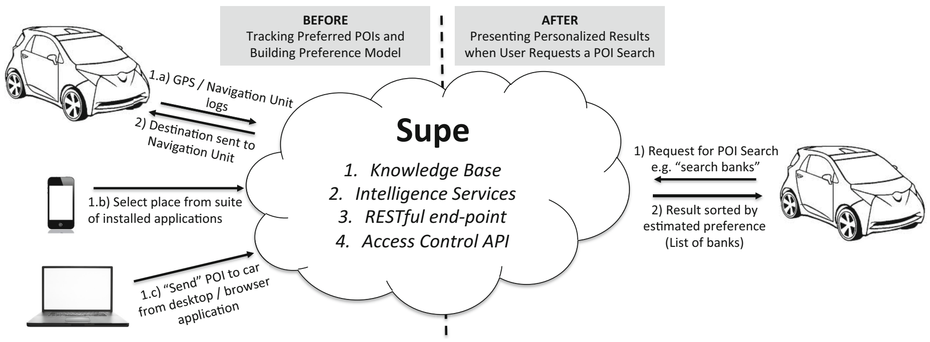

BEFORE
Tracking Preferred POIs and Building Preference Model
AFTER
Presenting Personalized Results when User Requests a POI Search
1.a) GPS / Navigation Unit logs
2) Destination sent to Navigation Unit
1.b) Select place from suite of installed applications
1.c) "Send" POI to car from desktop / browser application
Supe
1. Knowledge Base
2. Intelligence Services
3. RESTful end-point
4. Access Control API
1) Request for POI Search e.g. "search banks"
2) Result sorted by estimated preference (List of banks)

Fig. 1. Supe system Overview

understanding of place and driver data to an ontology. Supe also provides Intelligent Services, which use a machine learning engine in the background, for the presentation of the preferred places to the driver. Fig. 1 depicts the system overview.

The Semantic Web Knowledge System is at the heart of Supe and resides in the cloud. It contains a Knowledge Base, Intelligent Services, RESTful endpoints and access control mechanisms which are described in detail in Section 2.2. Supe is also connected to multiple devices that help collect POI data for the driver. Applications running on these devices use the exposed services to search and look-up places from multiple sources on the web. A place that the driver selects from the search results is accessible on the navigation system in his vehicle. For example, the driver might want to 'send' a POI from their desktop before getting into the car. Alternatively, they may select a place from their smart phones using their personal suite of installed applications and send it to the vehicle to be used as a destination for the in-vehicle navigation system. A third scenario finds places that the driver has visited using the history (GPS logs, etc.) of the in-vehicle head unit. In all three cases, Supe keeps track of the place consumed (i.e. navigated to, called, etc.) in order to add it to the driver's preferences. Once the preference model is built, it can be used to personalize the POI search in the in-vehicle navigation system. When the navigation system requests a place search, e.g. a bank, the system first retrieves POIs that match the search criteria, reorders these results according to the driver's preferences and returns the list to the navigation system.

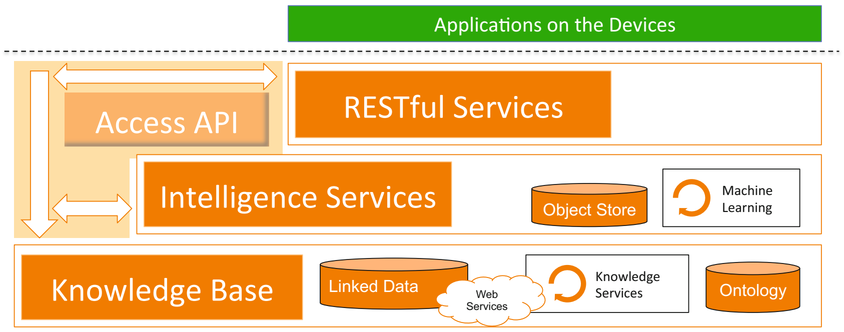


Fig. 2. Semantic Web Knowledge System Layer Cake

2.2 The Supe Semantic Web Knowledge System

Fig. 2 shows the architecture of Supe using a layer cake diagram. The components of the Semantic Web Knowledge System are discussed below.

Knowledge Base. The Supe Knowledge Base is based on Semantic Web and Linked Data standards. It is responsible for describing and storing the driver and place information. It also allows for using Web Services on-the-fly as a source for real-time place data in RDF. The Knowledge Base is composed of the following:

1. **Ontology:** Place-data in the system is represented using an RDFS ontology that covers the POI domain. The ontology defines a concept hierarchy of places based on their categories along with the properties associated with each of those place types. Fig. 3 shows a small subset of the ontology. It also has supporting concepts for representing user information, like home or work addresses, etc. All concepts and properties in the ontology belong to the *ontology namespace*.

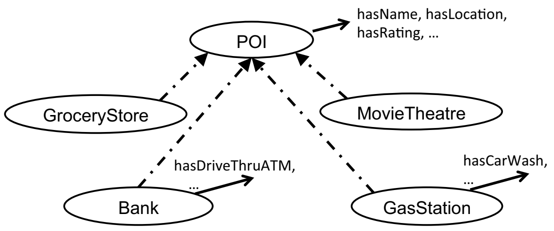


Fig. 3. Place Ontology (subset)

2. **Linked Data:** All instances for place and user data in the Knowledge Base are identified by URIs belonging to the *instance namespace* and are represented in RDF. Internally, the Intelligent Services can access this data as server objects that can be modified locally using the Jena framework [9]. Any data is also accessible in RDF or JSON [4] to the applications running on the devices for easy consumption as dereferencable URIs using the Linked Data Design Principles[2]. Locally, the RDF data for each instance is grouped together into an *instance molecule* and stored in a database with its URI as the primary key. We base our grouping of Linked Data triples into instance molecules by specializing the browse-able graph[2] terminology, where triples that have the URI for the instance in the object part of the triple are absent. Grouping triples into instance molecules allows us for a loosely coupled browse-able graph of data without duplicates. The search look-up is realized using a separate index that uses only the location and category. Since we are using Jena on the server side, any necessary RDFS inferences on the molecule can be performed when it is retrieved from the Knowledge Base. A similar concept to instance molecule, defined as *RDF molecule*, is also found in Ding et al.[5].
We also create a Linked Data wrapper around multiple sources on the web to find and retrieve POI instance data. For example, we can access each place already present on Yelp as Linked Data, as we wrap the Yelp API [1] and 'lift' the associated place data into our ontology. We use similar wrappers for other sources, like Google Places API[2], or data stored locally in an RDBMS store. Lifting all the data into the same ontology, allows us to integrate multiple sources and also create a richer set of information for modeling driver preference by merging such data.

[1] http://www.yelp.com/developers/getting_started/api_overview
[2] http://code.google.com/apis/maps/documentation/places/

3. **Knowledge Services:** If an instance molecule corresponds to an object and its attributes in an *object oriented programming language*, then Knowledge Services correspond to its functions. These services control what operations can be performed on the instance molecule (e.g. ensuring that the triples conform to the forward-only browse-able RDF graph terminology). A Knowledge Service can also build on other services. For example, the service that adds user and situation context to a place instance, uses the instance molecule modification knowledge service. Another use of Knowledge Services is to create more sophisticated data structures like lists, stacks, queues, etc. for the instances. For example, the POI Search service, given the location and category, uses one of the POI Web Services to search for places and then presents the result as a sorted list of instance molecules to the requesting Intelligent Services.

Intelligent Services: These are responsible for the intelligence in Supe. It contains the machine learning component that is used for modeling the driver's preferences and presenting a personalized search result. Since we use Linked Data to represent knowledge, the Intelligent Services need to convert RDF to data suitable for the machine learning algorithm used. The Intelligent Services are also supported with a database where the objects that it needs to persist, e.g. the driver's preference model, can be stored. These objects are also identified using URIs. The Intelligent Services for building preferences and personalizing search results are described in Section 2.3 and Section 2.4.

RESTful Services: POI data and Intelligent Servicesare exposed to client applications using RESTful end-points. Peripheral services for modifying user data (like home or work address, etc.) or services that facilitate the scenarios from Section 2.1 are also included. The end-points also know if these services are to be executed synchronously or asynchronously, depending on the whether the client device needs to wait on service output or not. For example, the client App does not have to wait while the preference model is getting updated with a new place instance.

Access Control API: Due to the highly personal nature of the data and to prevent RESTful services, Knowledge Services, Intelligent Services and applications running on the client devices (collectively referred to as *platform services* below) from corrupting other service or instance data, Supe also has an access control mechanism in place. Users and platform services use an identifier and a secret passkey combination for authentication. To simplify access control, we use an approach that restricts access based on namespaces rather than on individual instances. In order to access instances in the necessary namespace, an authentication challenge needs to be met successfully. Access control rules for the different namespaces are described below in brief:

1. **Ontology Namespace:** All platform services have access to the ontology.
2. **Instance Namespace:** All platform services have access to the Linked Data that is not user or service specific.

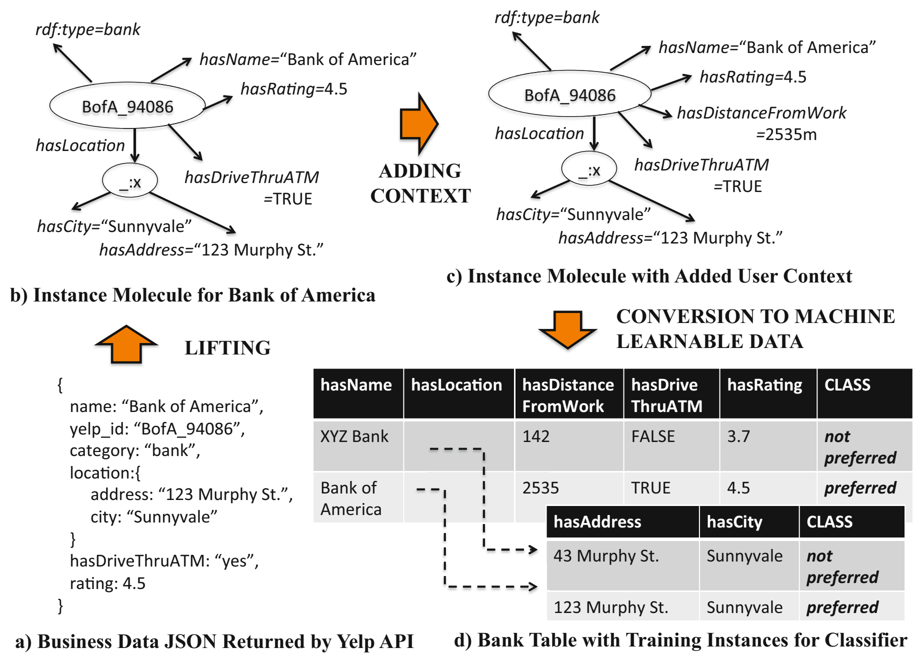
3. **Namespaces for each user:** All user specific data, like his home or work address, etc., that is not owned by any particular platform service belongs to his separate namespace. Access is granted to all platform services and client devices that can respond to user authentication challenge.
4. **Namespaces for each platform service:** Data belonging to a platform service but not specific to any user (e.g. service configuration, service static objects, etc.) belongs to this namespace. A platform service cannot access data in some other service's namespace.
5. **Subspaces for each platform service within a user's namespace:** All platform services with user specific data save it in a sub-space created for that service within the user's namespace. A platform service cannot access data belonging to a user that it does not have the necessary authentication for. Alternatively, even if it does have user authentication, it cannot access another service's data.

To access Linked Data or Intelligent Services internally, a platform service needs to pass the user and the service credentials to the Access Control API as function parameters. On the client side, any authentication parameters to be passed to the platform services are forwarded as HTTPS request query parameters (i.e. as a secured GET / POST query). HTTPS also ensures the privacy of the personal information transfer between the client applications and Supe. URIs in Supe start with 'https://' and are thus dereference-able as well as secure. Use of HTTPS relieves the application layer of the task of encryption of the data and makes the implementation of the system easier.

2.3 Building Driver Preferences from Tracked POI Data

When client applications send data to the in-vehicle navigation system, Supe tracks the visited/consumed POIs and stores them as driver history. This data is then used to build a statistical model of the driver's place preferences by training a machine learning algorithm. This process, is described below. Fig. 4 shows the steps involved in converting POI data about a bank, which the user selected in the navigation system, to machine learnable data.

Fetching Linked Data for POIs (Lifting): Identifiers of POIs that are tracked are used to build the preference model. Data for these places is retrieved from multiple web sources using the *POI Search knowledge service*. This returns place data in RDF using the ontology after lifting the web service response. For example, when the user selects a bank called "Bank of America", the lifting process converts the JSON response of the web service (e.g. Yelp API) into an instance molecule as shown in Fig. 4 (a). Lifting data to a single ontology also allows us to support multiple sources for the POI data. Different Web Services like Google Places API have different representations for data (e.g. different metrics, different category hierarchy, etc.). Search results from these sources can be integrated into the same instance molecule, using appropriate lifting logic.


rdf:type=bank
hasName="Bank of America"
hasRating=4.5
BofA_94086
hasLocation
_:x
hasDriveThruATM =TRUE
hasCity="Sunnyvale"
hasAddress="123 Murphy St."
b) Instance Molecule for Bank of America
ADDING CONTEXT
rdf:type=bank
hasName="Bank of America"
hasRating=4.5
BofA_94086
hasDistanceFromWork =2535m
hasLocation
_:x
hasDriveThruATM =TRUE
hasCity="Sunnyvale"
hasAddress="123 Murphy St."
c) Instance Molecule with Added User Context
LIFTING
CONVERSION TO MACHINE LEARNABLE DATA
a) Business Data JSON Returned by Yelp API
d) Bank Table with Training Instances for Classifier

```
{
  name: "Bank of America",
  yelp_id: "BofA_94086",
  category: "bank",
  location:{
     address: "123 Murphy St.",
     city: "Sunnyvale"
  }
  hasDriveThruATM: "yes",
  rating: 4.5
}
```

hasName	hasLocation	hasDistance FromWork	hasDrive ThruATM	hasRating	CLASS
XYZ Bank		142	FALSE	3.7	*not preferred*
Bank of America		2535	TRUE	4.5	*preferred*

hasAddress	hasCity	CLASS
43 Murphy St.	Sunnyvale	*not preferred*
123 Murphy St.	Sunnyvale	*preferred*

Fig. 4. Building User Preferences (Note: Only subset of the actual data used, is shown.)

Adding Context: A major design consideration for using an ontology for representation in Supe is to automatically extrapolate context that would help for a richer description of the data. The *Context knowledge service* is used to add user and situation context to POI instance molecule using rules programmed in the service. To add user data, it first loads the data for the driver from the local Linked Data store. Based on the programmed rules, it then automatically adds triples for the user context to the place data instance molecule. For example, if the user has his home or work address stored, then the service is able to add *distance from home* or *distance from work* context. As shown in Fig. 4 (b) and (c), user context *hasDistanceFromWork* is added to the Bank of America instance molecule. Another context that can be added is Situation. For example, information whether the visit was in the morning or evening along with the time spent can help understand the if the driver accessed an ATM or actually had to stop by at the bank for a longer time.

Conversion of RDF Instance Molecule to Machine Learnable Data: The instance molecule containing the POI and context data is used to learn the user's preferences using a content based approach. This instance needs to be first converted to a representation that machine learning algorithms can understand. The translation from Linked Data to machine learning instances is relatively straight-forward. We explain this translation as a conversion of instance molecules into a table containing training data, as used by conventional machine learning algorithms, below. (Due to lack of space, Fig. 4 (d) only shows a representative set of columns that can be derived from Fig. 4 (c)). The tables are implemented as frequency counts for values in each column and persisted in the object store. The learning algorithm used is described later, in Section 2.4.

1. **The Table:** All instances belonging to a certain category are grouped together in a single table. For example, Fig. 4 (d) shows a subset of the table for banks.
2. **Rows in the Table:** Each instance to be added to the preference model translates to each row in the table. The URI can either be dropped from the training (since row identifiers usually do not contribute to the learnt model) or can alternatively be added as a new column in the table. We use the latter approach to bias the model with a higher preference to a previously visited place [3].
3. **Columns in the table** The property-value pairs for the instance get translated as columns and values in the table. The properties for which the type of the table is a domain, appear as columns. For example, while the *hasDriveThruATM* is a property of Banks, the *hasName* property is inherited from the parent concept *POI* and is also present in the table in Fig. 4.
4. **Values in the Table:** RDF Literal values are translated as one of string, numeric or nominal values in a column. For example, the translation of the values for the *hasName* & *hasRating* properties from Fig. 4 (c). to columns in the table in Fig. 4 (d) is trivial. These translation rules can be specified explicitly in the ontology at construction time. Alternatively, a type detection mechanism could be used to identify the data types. For values of properties that are blank nodes, we use nesting of tables. This results in tracking inner frequency counts for the properties. Section 2.4 describes how the inner nested tables are used to compute preference. For values of properties that are URIs, we can choose to either (i) use the lexical value of the identifier as the value of the attribute in the machine learning table or (ii) represent the instance in a nested table, similar to blank nodes.
5. **Class Column in the Table:** The class value in the table for the training instances is marked as either 'preferred' or 'notpreferred' based on the way the data is tracked. The POIs in the driver history of visited places are marked as preferred (e.g. Bank of America is marked as *preferred* in the class column in Fig. 4 (d)).

Incrementally Building and Storing the Model: Instead of building the entire preference model from scratch every time a new instance is added, we use an incremental approach to building the model. Each time a new place is to be added to this model, the stored model is retrieved from the object store, updated and persisted back into the object store. To make sure that the performance doesn't start degrading once the size of the model becomes large, it only contains the n-most recent preferred places. Optimization is also made to reduce disk writes by using a delayed write of the preference model object.

[3] This column is not depicted in Fig. 4 (d) due to lack of space.

2.4 Ordering POIs According to User's Preference

Once the preference model contains some POI data, applications requesting a search of POIs can be presented with personalized results. In our use-case, the user might want to search for banks around him using the in-vehicle navigation system, when in a place that he does not know much about. The steps involved in realizing this are part of the *Personalized POI Search Intelligent Service*, and are described below.

Fetching POIs and Adding Context: The personalized POI search service uses the POI search knowledge service to retrieve places that match the search criteria from web sources. Necessary user and situation context are added to the list of instances, which were already lifted into RDF by the knowledge service, if necessary data is present. These two steps are similar to the steps described in Section 2.3. The search service then uses the Preference Scoring service to estimate the user's affinity to each place.

Scoring Each POI Using the Stored Preference Model: The user's preference model is loaded by the Preference Scoring Intelligent Service from the object store. Each place instance it receives is first converted into a form suitable for input to scoring by the machine learning algorithm. For calculating the preference of a POI to the user, we use a scoring function that computes the Euclidean distance, in a way similar to the nearest neighbor calculation in unsupervised classification algorithms. The scores for the POIs are returned to the search service. It can then sort the list of POIs according to the score and present the personalized list to the requesting client application.

***Distance-from-Unity* Metric:** We introduce a similarity metric for preference that uses a 'nearest neighbor' approach to calculate the most preferred place. For an instance that is to be scored, we project the the likelihood probabilities for the values of its attributes on orthogonal axes. For a value of a property, the likelihood probability is calculated as $P(\text{'}value' \mid preferred)$ based on the frequency count table of that column. On each axis, the likelihood probability gets projected as a continuous numeric function with maximum possible score of 1.0 for a value that is always preferred, and a score of 0.0 for a value that is absent from the table. The instance gets projected as a point in this multi-dimensional space. A hypothetical instance that will always be preferred will have its likelihood value 1.0 on all axes. The point representing this hypothetical instance will have the co-ordinates (1.0, 1.0, 1.0, 1.0, 1.0 ...) in the multidimensional space. We call this point as *'Unity'*. For the instance that we want to score, we calculate the Euclidean distance of the projected point from Unity. The personalized search service can then sort the list of POIs according to this distance such that the instance whose distance is minimum is considered most preferred.

For literal values the probability is calculated as follows: (i) probabilities for numeric columns are calculated using a Gaussian distribution, (ii) probabilities for string values are calculated similar to a naïve-Bayes probability based document similarity metric and (iii) probabilities for nominal values are calculated as the probability of the symbol occuring in the column. In case of numeric attributes, where we use a Gaussian Distribution, we normalize the probability density function by dividing it with the probability density of the mean. This puts the probability 'distance' between zero and one. To calculate the distance for object values for an attribute, we explode that attribute into its own hyperspace. We calculate the distance of the inner objects to the unity point in this hyperspace. Dividing this distance with the distance from origin to unity in that hyperspace normalizes it to a value between 0 and 1. The hyperspace for the attribute is then collapsed back to a single axis and the normalized distance value becomes distance from unity on that axis, in the original multi-dimensional space. For a multivalued property, we take the average of the distances of all its values. If we were to score the affinity for Bank of America from Fig. 4 (d) for testing, the calculation would be as follows.

$$D(BofA\_94086) = \sqrt{\begin{array}{r} (1 - P(\text{“}Bank\ of\ America\text{”} \mid preferred))^2 \\ +(1 - P(2353 \mid preferred))^2 + (1 - P(TRUE \mid preferred))^2 \\ +(1 - P(4.5 \mid preferred))^2 + D(\_ : x)^2 \end{array}}$$

$$Where\ D(\_ : x) = \sqrt{\frac{\begin{array}{r} (1 - P(\text{“}Sunnyvale\text{”} \mid preferred))^2 \\ +(1 - P(\text{“}123\ Murphy\ St.\text{”} \mid preferred))^2 \end{array}}{2}}$$

Naïve-Bayes Probability: To establish a baseline for comparison, we also use a second scoring function based on naïve-Bayes classification algorithms. For an instance to be scored, we calculate the navë-Bayes probability of it being preferred. For two instances that are normalized, we can compare the probabilities of each of them belonging to the class 'preferred' to decide which one of them is more likely to be preferred by the user. The probabilities can be calculated easily by using the frequency count tables for the columns. The calculation of the probability for Bank of America from Fig. 4 (d) is shown below.

$$\begin{aligned} P(preferred \mid BofA\_94086) = P(preferred) & \\ & \times \frac{P(\text{“}Bank\ of\ America\text{”} \mid preferred)}{P(\text{“}Bank\ of\ America\text{”})} \\ & \times \frac{P(2353 \mid preferred)}{P(2353)} \times \frac{P(TRUE \mid preferred)}{P(TRUE)} \\ & \times \frac{P(4.5 \mid preferred)}{P(4.5)} \times \frac{P(\_{:}x \mid preferred)}{P(\_{:}x)} \end{aligned}$$

$$\begin{aligned} Where \frac{P(\_{:}x \mid preferred)}{P(\_{:}x)} = & \frac{P(\text{“}Sunnyvale\text{”} \mid preferred)}{P(\text{“}Sunnyvale\text{”})} \\ & \times \frac{P(\text{“}123\ Murphy\ St.\text{”} \mid preferred)}{P(\text{“}123\ Murphy\ St.\text{”})} \end{aligned}$$

While using naïve-Bayes however, we take three important precautions. First, we also populate the table with negative training examples so that the fractions do not cancel each other out. In the absence of explicit information about the places that the driver dislikes, we train the model with instances, other than the one that was selected while performing the POI search, as belonging to the class *not preferred*. Second, unlike conventional classification, where the probability of an instance belonging to a certain class is compared with its probability in belonging to other classes, we compare the probability of one instance being preferred to the probability of another instance being preferred. This demands that we normalize all the instances (i.e. have the same set of properties and use the same frequency tables for calculating the probabilities) by ensuring that all instances have the same set of attributes by either assigning default values or dropping missing attributes for all instances. Third, in case of multivalued attributes, we take averages for the score of each value for ensuring that the scores are normalized.

3 Evaluation and Discussion

We first describe an initial evaluation of the scoring mechanisms using dummy users and then discuss the preliminary findings of an actual user study, currently in progress. To see if the scoring functions work, we constructed an experiment to build and test a preference model for dummy users. A dummy-user is a computer program that is able to perform selections of POIs based on an embedded user profile. Each (dummy) user performs a POI search task for places belonging to a category (e.g. restaurants, banks, grocery stores, etc.) and programmatically selects one POI (or multiple correct POIs) based on its profile. The profiles for these users are listed below:

1. uNearest: Always selects the nearest place.
2. uHighestRated: Always selects the highest rated place.
3. uBrand: Always selects certain brands like Starbucks, Chevron, etc.
4. uCategory: Always selects places belonging to certain sub-categories (e.g. Japanese restaurants, etc.)
5. uYelpOrder: Always trusts the system and selects the first item. In this case the list is ordered by the underlying Web Service(Yelp) that sorts places using a proprietary special mix.
6. uPerson: This user was modelled based on the preferences of a real person after asking him a set of questions and then programming his choices into the dummy user.

Training was done on 50 tasks in an area near the user's routine commute. For example, some of the search use-cases were 'find a gas station near home', 'find a restaurant for lunch near work', etc. Following this, testing was done on a set of 25 POI search tasks in an unknown location along with the fifty examples from the training set. The testing tasks were formulated on hypothetical use-cases of likely requests by the user in a new area. For example, some of the use-cases were 'find a restaurant near the airport (when the user picks a friend up)', "Find a coffee shop in Santa Cruz while driving San Francisco for a break", etc. We compared the two scoring functions where the accuracy was calculated as follows. For the list of places returned sorted according to either scoring mechanism, if the dummy user selection would have matched the first place, we count this as the preference modeling for the task as a success. Accuracy represents the % of successful tasks. The results of the experiment are shown below in Fig. 5. These two scoring mechanisms were selected for (i) checking that the preferences worked by manually checking 'under-the-hood' of the learnt models, and (ii) establishing a baseline for incorporating more sophisticated scoring techniques in the future.

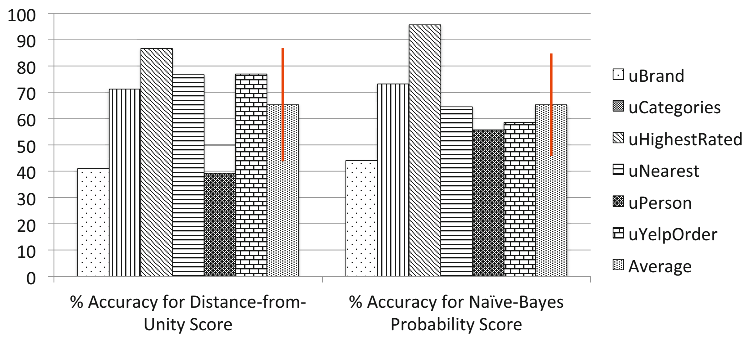
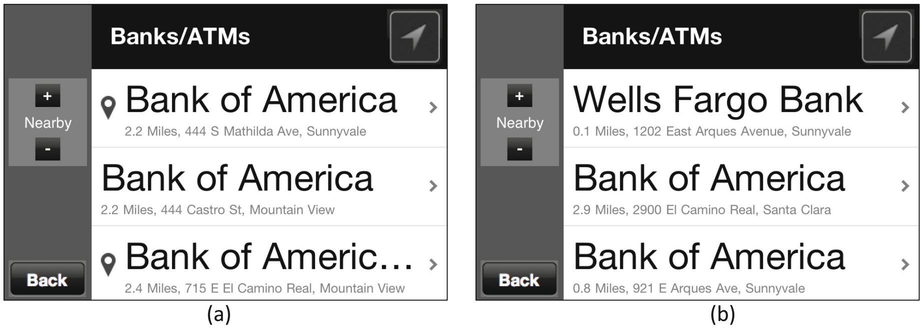


Fig. 5. Accuracy of Scoring Functions for Dummy Users

As these results were promising, we implemented a smart-phone application emulating the in-vehicle navigation system using client and server side implementations for a user study. We emulated the in-vehicle head unit by using a smart-phone app that allows users to search for POIs belonging to any category, while the cloud based server was implemented using the description of Supe above. Though it is still to early to provide experimental results on the effectiveness of the preference model for actual users, the following screenshot (Fig. 6) shows one case where a user found the preference model to work effectively. The first image shows the results for banks in the user's daily commute area. The ones marked with a pin are the places that he has visited and are used to build his preference model. When the user was away from his regular zone and in need of cash, he searched for banks using the POI Search service. The second capture shows that two of his preferred banks were ranked among the top three in the list. Intuitively the reader may figure out that this user prefers a specific banking company. The model is able to detect this preference because the likelihood probability on the *hasName* axis is high. We are finding similar patterns in other users as well for other categories, like gas stations, restaurants, etc. and hope to present these results at the conference. The *Distance-from-Unity* metric was used to find the results shown in the screenshots. While the naïve-Bayes scoring function outperforms the *Distance-from-Unity* scoring function in the preliminary experiments, in practice users rarely have a rigid choice pattern like always selecting the nearest place. The preliminary experiment relied on the order information on properties like distance, rating, etc. to figure out these preferences.


(a)

Banks/ATMs

Nearby

Bank of America
2.2 Miles, 444 S Mathilda Ave, Sunnyvale

Bank of America
2.2 Miles, 444 Castro St, Mountain View

Bank of Americ...
2.4 Miles, 715 E El Camino Real, Mountain View

Back

(b)

Banks/ATMs

Nearby

Wells Fargo Bank
0.1 Miles, 1202 East Arques Avenue, Sunnyvale

Bank of America
2.9 Miles, 2900 El Camino Real, Santa Clara

Bank of America
0.8 Miles, 921 E Arques Ave, Sunnyvale

Back

Fig. 6. Screenshots of the Implemented User Study Application: Search result for banks (a) in daily commute area (places previously visited are marked) (b) in a new area using Distance-from-Unity metric

In the absence of such explicit input, the *Distance-from-Unity* function was able to better hone in the driver's preferences than the naïve-Bayes scoring function and was chosen to personalize the search results.

4 Related Work

Before we discuss similar work that create content based preference models for users, we briefly describe other work related to parts of our system, which have been explored in the Semantic Web. The POI Search Knowledge Services that we define are wrappers around existing Web Services and produce RDF as output similar to Semantic Web Services[10]. They borrow the 'lifting' concept from work in Semantically Annotated WSDL services[7], where the output of the web service is 'lifted' from XML into a semantic model. In our case, we have programmed the logic for lifting in our different Knowledge Services that may choose from different sources like Yelp, Google Places for place data, since the domain is fairly constant. But there are other solutions (e.g. Ambite et. al[1]) that can be alternatively used to automatically find alignments between sources and construct Semantic Web Services. Though access control mechanisms for the Semantic Web have been explored for a long time, it has only actively been researched for the Linked Data more recently. Hollenbach et. al [6] describe a decentralized approach to access control for Linked Data at the document level. Wagner et. al [14] describe how policies can be implemented for controlling access to private Linked Data in the semantic smart grid. Our approach for controlling access does not include RDF descriptions of authentication and authorization credentials but is instead based on existing practices for Web Services which allow for easy integration with client devices. It is also easier to implement.

In the past year, the combination of RDF and machine learning has gained some traction. The work that is perhaps most similar to our approach on converting RDF to a machine learning table, is found in Lin et. al [8]. For the movie domain, they try to predict if a movie receives more than $2M in its opening week by converting the RDF graph for the movie to a Relational Bayesian Classifier. One major difference in their approach to ours is the translation of multi-valued object properties (e.g. hasActor) into the relational table. While doing so, their approach 'flattens' all objects and groups values into properties. For example, names of all actors get aggregated into a single 'has actor name' column and all actors' year of birth get aggregated into a 'has actor YoB' column. In doing so, the associativity within an instance (e.g. of an actor's name to his age) is lost. In our approach, the advantage of using a graph for instance representation which is the associativity between the properties is maintained since (i) the inner objects are scored independently and (ii) in case of multi-valued object properties, we take an average of individual scores. However, a more in-depth comparison of the two approaches needs to be conducted on common data for better analysis. Another related work in learning from RDF has been explored in Bicer et. al[3], where relational kernel machines are used for movie recommendations. Content based preference modeling has received large research attention in recommendation systems over the past few years[13]. These algorithms, use machine learning techniques like linear regression, naïve-Bayes, etc. for finding recommendations. For example, the Syskill & Webert system[12] uses a naïve Bayes-classifier for classifying web sites as either 'hot' or 'cold'. Similar to our approach, this system also uses the probability score to rank pages according to user's preferences. More recently, personalized information retrieval has also been gaining traction in the Semantic Web. For example, *dbrec*[11] is a music recommendation system based on DBpedia that uses a semantic link distance metric for its recommendations.

5 Conclusion and Future Work

In this paper we have described the architecture and implementation of a system that combines the Semantic Web, a Linked Data Knowledge Base, and machine learning to build a preference model of the places that a driver likes using a content-based approach. The use of a Semantic Web Knowledge System that combines knowledge and intelligence has advantages. First, it helps represent the preference model in a way that can be consistently interpreted by knowledge services as well as intelligent services. Second, data about places, user, etc can be enriched by integrating it from multiple sources, along with extrapolating user and situation context. Lastly, this RDF data can automatically be translated into a format compatible with machine learning algorithms for building the preference model. Preliminary results from our ongoing user study show that the preference model is able to predict the driver's estimated affinity to a place. It can thus be used for POI recommendation in the vehicle, where we would like to minimize the driver interaction by providing a personalized experience.

Though our preliminary evaluation shows promising results with actual users for using simple metrics, it has scope for improvements. Since a major focus of the study is to understand the preference model and improve its accuracy, we intend to explore other machine learning techniques (e.g. linear regression) and feature selection techniques to improve the preference model, as future work. During the course of the user study, we are also dealing with performance issues of implementing a Semantic Web Knowledge System in the cloud, e.g. the overhead associated with integrating from multiple sources, adding contexts, RDF to machine learning translations, etc. Optimizing this performance would prove as a great learning opportunity. Lastly, another direction for future work is in using the system for exploring other domains like music, navigation, etc. for personalization in the vehicle. We believe that Supe has the potential of graduating from an in-use intelligent system for studying and understanding driver preferences of POIs to a real world deployment that can provide a comprehensive personalized experience in the car.

References

1. Ambite, J.L., Darbha, S., Goel, A., Knoblock, C.A., Lerman, K., Parundekar, R., Russ, T.: Automatically Constructing Semantic Web Services from Online Sources. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 17–32. Springer, Heidelberg (2009)
2. Berners-Lee, T.: Design issues: Linked data (2006), `http://www.w3.org/DesignIssues/LinkedData.html`
3. Bicer, V., Tran, T., Gossen, A.: Relational Kernel Machines for Learning from Graph-Structured RDF Data. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part I. LNCS, vol. 6643, pp. 47–62. Springer, Heidelberg (2011)
4. Crockford, D.: The application/json media type for javascript object notation, json (2006), `https://tools.ietf.org/html/rfc4627`
5. Ding, L., Finin, T., Peng, Y., Da Silva, P., McGuinness, D.: Tracking RDF graph provenance using RDF molecules. In: Proc. of the 4th International Semantic Web Conference, Poster (2005)
6. Hollenbach, J., Presbrey, J., Berners-Lee, T.: Using rdf metadata to enable access control on the social semantic web. In: Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (2009)
7. Kopeckỳ, J., Vitvar, T., Bournez, C., Farrell, J.: Sawsdl: Semantic annotations for wsdl and xml schema. IEEE Internet Computing, 60–67 (2007)
8. Lin, H.T., Koul, N., Honavar, V.: Learning Relational Bayesian Classifiers from RDF Data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 389–404. Springer, Heidelberg (2011)
9. McBride, B.: Jena: A semantic web toolkit. IEEE Internet Computing 6(6), 55–59 (2002)
10. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic web services. IEEE Intelligent Systems 16(2), 46–53 (2001)
11. Passant, A.: dbrec — Music Recommendations Using DBpedia. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part II. LNCS, vol. 6497, pp. 209–224. Springer, Heidelberg (2010)
12. Pazzani, M., Billsus, D.: Learning and revising user profiles: The identification of interesting web sites. Machine learning 27(3), 313–331 (1997)
13. Pazzani, M.J., Billsus, D.: Content-Based Recommendation Systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) Adaptive Web 2007. LNCS, vol. 4321, pp. 325–341. Springer, Heidelberg (2007)
14. Wagner, A., Speiser, S., Raabe, O., Harth, A.: Linked data for a privacy-aware smart grid. In: INFORMATIK 2010 Workshop-Informatik für die Energiesysteme der Zukunft (2010)