Learning Consumer Preferences for Content-Oriented Negotiation *

Reyhan Aydoğan
reyhan.aydogan@gmail.com

Pınar Yolum
pinar.yolum@boun.edu.tr

Department of Computer Engineering
Boğaziçi University
Bebek, 34342, Istanbul,Turkey

ABSTRACT

Current approaches to e-commerce treat service price as the primary construct for negotiation. However, negotiation on price presupposes that other properties of the service have already been agreed upon. For many real life situations, a service consumer and a producer has a difficult time in understanding each other's requirements and forming a consensus on the desired service. Accordingly, this paper develops a theory for automating service negotiation and provides the details of an implemented system. The theory combines important ideas from inductive logic learning with expressive representation of ontologies. Both consumers and producers share an ontology about the service of interest. Through repetitive interactions, producers learn consumers' needs accurately and can make better targeted counter offers. The developed system uses the well-known Wine ontology to demonstrate the negotiation of service needs.

1. INTRODUCTION

Traditional e-commerce applications are targeted for human users. However, as the number and extent of transactions increase, there is a tremendous demand for developing e-commerce applications that can be flexibly used by machines. Agents have proven to be successful paradigm for autonomous and intelligent software such as those needed for flexible e-commerce applications. In a typical e-commerce application, there are two basic but important roles: producer and consumer. Producer advertises and provides a service for which the consumer has interest, whereas the consumer requests and possibly accepts the service. In this case, service can be selling a book, reserving a hotel room, and so on.

Many times the service provider may not be offering the exact requested service due to lack of resources, constraints in its business policy, and so on. When this is the case, the producer and the consumer need to negotiate the *content* of the requested service [15]. Most of the current studies related to negotiation assume that the service content is fixed and thus focus on the price of the service [10]. Other approaches assume that all parts of the service are equally important and hence negotiate each service feature one by one [7, 3]. However, usually each service feature is not equally importance for a consumer. Moreover, importance of a service feature may vary from consumer to consumer. For instance, completion time of a service may be important for one consumer whereas the quality of the service may be more important for a second consumer. Thus, if a provider can learn the preferences of a consumer, it can provide counter offers that are better targeted for that particular consumer.

As an alternative to price-oriented negotiation approaches, we propose an approach for automating the content negotiation of services, where the producers learn the preferences of consumers over time. We represent service requests as a vector of service features. We use an ontology [8, 4] in order to capture the relations between features and to construct the features for a given service. By using a common ontology, we enable the consumers and producers to share a common vocabulary for negotiation. The producer models the requests of the consumer and its counter offers to understand which features are more important for the consumer. To enable this learning, we need an incremental learning approach that can be trained at run time and can revise itself with each new example. We have used inductive learning [11] for this purpose and extended the version space algorithms to take into account the needs of service negotiation context.

The rest of this paper is organized as follows: Section 2 explains our proposed architecture. Section 3 gives the technical details of our approach including the algorithms for learning consumers' service needs and offering services accordingly. Section 4 provides our experimental setup and our test case. Section 6 discusses and compares our work with other related work.

2. ARCHITECTURE

Our main components are consumer and producer agents, which communicate with each other to perform content-oriented negotiation. To make the most of the communication, two requirements have to be fulfilled. One, the language the agents speak should be free from ambiguities. Two, the language should be expressive enough for parties to understand and negotiate details. To achieve both of these requirements, we use ontologies in our architecture. A shared ontology provides the necessary vocabulary for agents and hence enables a common language for agents. Further, since an ontology can represent concepts, their properties and their relationships semantically, the agents can reason the details of the service

*This research is supported by Bogazici University Research Fund under grant BAP05A104. We thank the anonymous reviewers for helpful comments.

that is being negotiated. Our architecture is independent of the ontology used. However, to make our discussion concrete, we use the well-known Wine ontology [18] to illustrate our ideas and to test our system. The wine ontology includes features such as color, body, winery of the wine and so on. With this ontology, the service that is being negotiated between the consumer and the producer is that of wine.

Figure 1 depicts our architecture. The consumer agent represents the customer and hence has access to the preferences of the customer. The consumer agent negotiates with the producer based on these preferences. Similarly, the producer agent has access to the producer's inventory and knows which wines are available or not. The inventory information is represented as a set of individual classes in the ontology.

The architecture advances in a turn-taking fashion, where the consumer agent starts the negotiation with a particular service request. The request is composed of significant features of the service. In the wine example, these features include as color, winery and so on. This is the particular wine that the customer is interested in purchasing. If the producer has the requested wine in its inventory, the negotiation ends. Otherwise, the producer offers an alternative wine from the inventory. When the consumer receives a counter offer from the producer, it will evaluate it. If it is acceptable, then the negotiation will end. Otherwise, the customer will generate a new request (or stick to the previous request). This process will continue until some service is accepted by the consumer agent or all the possible offer are put forward to the consumer by the producer.

3. TECHNICAL DETAILS

One of the crucial challenges of the content-oriented negotiation is the automatic generation of counter offers by the service producer. When the producer constructs its offer, it should consider three important things: the current request, consumer preferences and the producer's available services. Both the consumer's current request and the producer's own available service are accessible by the producer. However, the consumer's preferences in most cases will not be. Hence, the producer will have to understand the needs of the consumer from their interactions and generate a counter offer that is likely to be accepted by the consumer. This challenge can be studied in three stages:

- Representation: How do we represent the requests and counter offers of the producer so that they can reason about each other's actions?
- Learning: How can the agents learn about each other's preferences based on requests and counter offers?
- Revision: How do the agents revise their requests or offers based on incoming information?

3.1 Representation

The requests of the consumer and the counter offers of the producer are represented as vectors, where each element in the vector corresponds to the value of a feature. For instance, the customer may prefer medium and strong red wine. This can be represented with the following vector query [Medium, Strong, Red]. Or, if the customer prefers light and delicate white wine, this can be represented with the following query [Light, Delicate, White].

3.2 Learning Phase

The producer agent tries to learn the consumer's preferences using information obtained from the dialogues between the producer and the consumer. The preferences denote the relative importance degree of the features of the services demanded by the consumer agents. For instance, the color of the wine may be so important that the consumer insists on buying the wine whose color is "red" and rejects all the offers involving the wine whose color is "white" or "rose". On the contrary, the winery may not be so important as much as the color for this customer, so the consumer may have a tendency to accept any offer, which meets the color requirement, whose winery is different from the specified one in the request.

To tackle this problem, we propose to use inductive learning. This technique is applied to learn the preferences as concepts. Version Space [11] is one of the inductive learning approaches that learns concepts from observed examples. The aim of version space algorithm is to generate general rules or hypotheses, which involve the positive examples and do not include any negative examples. In fact, this general hypothesis represents the desired concept. To understand how the Version Space approach works, we investigate the Candidate Elimination Algorithm (CEA) that implements the Version Space. Algorithm 1 gives the pseudo code [12].

Algorithm 1 Candidate Elimination Algorithm

1: $G \leftarrow$the set of maximally general hypotheses in H
2: $S \leftarrow$the set of maximally specific hypotheses in H
3: For each training example, d
4: **if** d is a positive example **then**
5: Remove all hypotheses in G do not cover d
6: For each hypothesis s in S does not cover d
7: – Remove s from S
8: – Add to S all minimal generalizations h of s such that h covers d and some members of G are more general than h
9: – Remove from S any hypothesis that is more general than another hypothesis in S
10: **end if**
11: **if** d is a negative example **then**
12: Remove all hypotheses in S that cover d
13: For each hypothesis g in G does cover d
14: – Remove g from G
15: – Add to G all minimal specifications h of g such that h does not cover d and some members of S are more specific than h
16: – Remove from G any hypothesis that is less general than another hypothesis in G
17: **end if**

There are two kinds of sets representing the most general hypotheses (set G) and the most specific hypotheses (set S). These sets are used to capture the limits of the learned concept. That is, a concept is at least as specific as the hypothesis that exist in set S and at most as general as the hypothesis contained in set G. The boundaries of the most general hypotheses are as large as possible in that they cover the entire positive training example and possible positive instances; whereas those of the most specific hypotheses are as small as possible they minimally cover the observed positive samples. By using the most general hypotheses and the most specific hypotheses, we can produce all possible hypotheses for the target concept. As the algorithm receives positive and negative examples, it revises the G and S such that eventually G and S intersect in which case the concept has been learned.

At the beginning of the algorithm, G is initialized with the set of maximally general hypotheses that cover everything, whereas S is set with the set of maximally specific ones. During the interactions, each request of the consumer can be considered as a positive example and each counter offer generated by the producer and rejected by the consumer agent can be thought of as a negative exam-

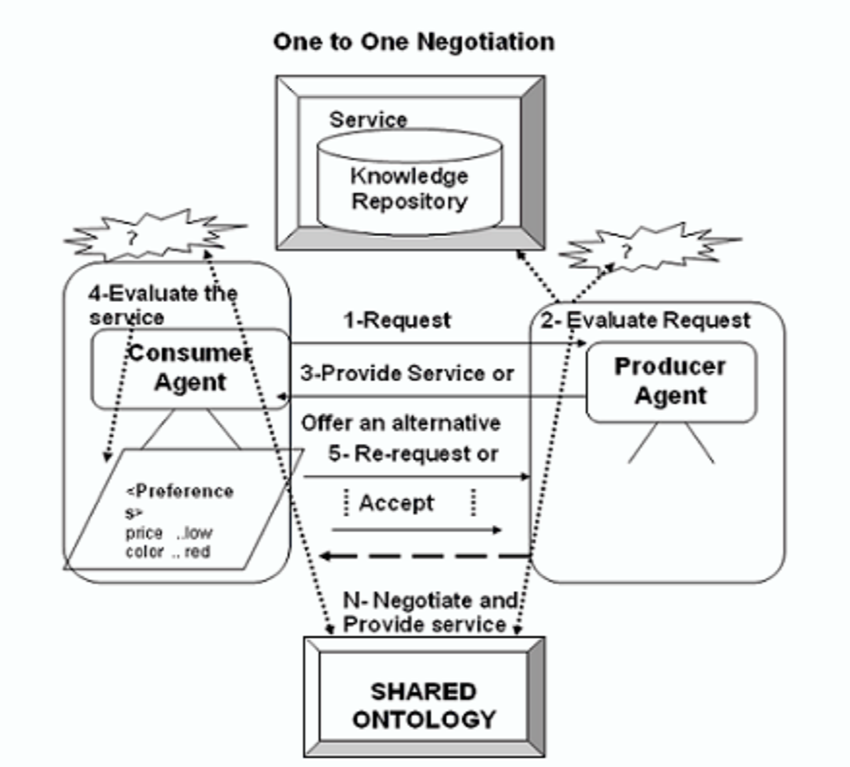

Figure 1: Negotiation Architecture

ple. At each dialogue between the producer and the consumer, we apply training over sets of the most specific and the most general hypotheses.

EXAMPLE 1. *Assume that our target concept, wine, has three attributes: body, flavor and color.*[1] *According to the scenario, assume a wine with features (Full, Strong, White) is an acceptable wine. Initially, G is simply initialized with the most general hypothesis, (?, ?, ?) where ? means this attribute can take any value. On the other hand, S is initialized with the most specific hypothesis consistent with the first positive observed training sample, in this case (Full, Strong, White).*

The negative samples enforce the specialization of some hypotheses so that G does not cover any hypothesis accepting the negative samples as positive.

EXAMPLE 2. *Following the previous example, assume that (Full, Delicate, Rose) is an unacceptable wine (i.e., a negative training sample). The general set should be specialized in such a way that it should not cover this instance but covers previously observed positive samples.*

In the algorithm, there may be various ways to specialize the general set G. One way is to compare the attributes of the negative example with the specific set and to assign their difference to the existing G. Also the algorithm eliminates the hypotheses that cover the negative sample from S.

EXAMPLE 3. *Assume that G contains (?,?,?) and S contains (Full, Strong, White). If the negative example (Full, Delicate, Rose) comes in, then G needs to be specialized such that it becomes (?, Strong, ?) and (?, ?, White).*

When a positive sample comes, the most specific set S should be generalized in order to cover the new training instance. Similarly, the details of the generalization is not specified in CEA; but from the illustration explained in [12], it can be performed by replacing the different values between special set and current positive sample with ?. Further, the most general hypotheses in G that do not cover the new positive sample are eliminated.

EXAMPLE 4. *Assume that S contains (Full, Delicate, Rose). If the next positive example is (Medium, Delicate, Rose), then the generalization of specific set S will be (?, Delicate, Rose).*

As a result, the most general hypotheses and the most special hypotheses cover all positive training samples but do not cover any negative ones. Incrementally, G specializes and S generalizes until G and S are equal to each other. When these sets are equal, the algorithm converges by means of reaching the target concept.

Algorithm 1 is primarily targeted for conjunctive concept. On the other hand, we need to learn disjunctive concepts in the negotiation of a service. For instance, a consumer may want strong white wine or delicate red wine but not want delicate white or moderate red wine. In Table 1, the behavior of the algorithm is shown clearly in accordance with the given samples. It can be seen that the algorithm fails in learning concepts where disjunctive requests exist. However, learning from such concepts is required when providing service to the consumer since consumer may have several alternative wishes.

The way the general set G is specialized and the special set S is generalized is crucial for successfully learning the hypothesis. Consider the following example:

[1] Wine class has seven attributes in our implementation in Section 4. In order to keep the examples simple, here we only include three attributes.

Table 1: When Candidate Elimination Algorithm fails

Sample Type	Sample	The most general set	The most specific set
+	(Full,Strong,White)	{ (?, ?, ?) }	{ (Full,Strong,White) }
-	(Full,Delicate,Rose)	{ (?, Strong, ?),(?, ?, White) }	{ (Full,Strong,White) }
+	(Medium,Moderate,Red)	{ }	{ (?, ?, ?) }

EXAMPLE 5. *The color feature of the wine product is allowed to take three different values: white, red and rose. Assume the first positive sample is (Light, Delicate, Red), denoting a red, light and delicate wine. And, the second positive sample is (Light, Delicate, White), denoting a white, light and delicate wine. According to Algorithm 1, the special set S will be generalized to be (Light, Delicate,?). However, now S also contains samples for (Light, Delicate, Rose) even though the consumer has not shown any interest in rose wine so far. If the consumer now rejects an offer of rose, light and delicate wine, the special set will be in violation with the consumer's preferences and the algorithm will end up learning the preferences incorrectly.*

Algorithm 2 Modified Candidate Elimination Algorithm

```
1: G ←the set of maximally general hypotheses in H
2: S ←the set of maximally specific hypotheses in H
3: For each training example, d
4: if d is a positive example then
5:    Add d to S
6:    if s in S can be combined with d to make one element then
7:       Combine s and d into sd {sd is the rule covers s and d}
8:    end if
9: end if
10: if d is a negative example then
11:    For each hypothesis g in G does cover d
12:    * Assume : g = (x1, x2, ..., xn) and d = (d1, d2, ..., dn)
13:    – Remove g from G
14:    – Add hypotheses g1, g2, gn where g1= (x1-d1, x2,..., xn),
       g2= (x1, x2-d2,..., xn),..., and gn= (x1, x2,..., xn-dn)
15:    – Remove from G any hypothesis that is less general than
       another hypothesis in G
16: end if
```

We deal with this problem by extending our hypothesis language to include disjunctive hypothesis in addition to the conjunctives and negation. Each attribute of the hypothesis has two parts: inclusive list which holds the list of valid values for that attribute and exclusive list which is the list of values which cannot be taken for that feature. For instance, if the most specific set is { (Light, Delicate, Red)}, it will be {(Light, Delicate, [White, Red])}instead of (Light, Delicate,?) after a positive example, (Light, Delicate, White) comes. Only when all the values exist in the list, they will be replaced by ?.

We modify the CEA algorithm (Algorithm 1)to deal with this change. The modified algorithm is given in Algorithm 2. The initialization phase is same with the original one (Lines 1, 2). If any positive sample comes, we add the sample to the special set as before (Line 4). However, we do not eliminate the hypotheses in G that do not cover this sample since G now contains a disjunction of many hypotheses, some of which will be conflicting with each other. Removing a specific hypothesis from G will result in loss of information, since other hypotheses are not guaranteed to cover it. After a time, some hypotheses in S can be merged and can construct one hypothesis (Lines 6, 7).

When a negative sample comes, we do not change S as before. We only make the most general hypotheses not cover this negative sample (Line 11–15).

EXAMPLE 6. *Table 2 illustrates an example that works with the modified version space algorithm. The initial request of the consumer is same as the one in Table 1, so as the initial G and S. When the counter offer from the producer comes (negative instance), we specialize G since G should not cover any negative instances. We replace (?, ?,?) by three disjunctive hypotheses; each hypothesis being minimally specialized. In this process, at each time one attribute value of negative sample is applied to the hypothesis in the general set.*

Note that in Example 6, we do not eliminate {(?-Full), ?, ?} from the general set while having a positive sample such as (Full, Strong, White). This stems from the possibility of using this rule in the generation of other hypotheses. For instance, if the example continues with a negative sample (Full, Strong, Red), we can specialize the previous rule such as {(?-Full), ?, (?-Red)}.

By Algorithm 2, we do not miss any information. Later, we plan to integrate ontology reasoning into the algorithm so that hierarchical information can be learned from subsumption hierarchy object or subclass of relations. Further, by using relationships among features, the producer can discover new knowledge from the existing knowledge.

EXAMPLE 7. *Using the wine ontology the following reasoning can be made: Bordeaux is defined as a Wine, Medoc is defined as a Bordeaux and Pauillac is defined as a Medoc. When the consumer agent wants to buy wine, the producer agent can offer any instance of Bordeaux, Medoc and Pauillac since it can reason that if Medoc is a Bordeaux and Bordeaux is a Wine then Medoc is a Wine and then if Pauillac is a Medoc and Medoc is a Wine then Pauillac is a wine. Such reasoning makes our agent use the semantic information related to the concepts.*

3.3 Offering Service

Making the right service offers is the most important process in content-oriented negotiation. To generate the best offer, the producer agent uses its service ontology and the inductive learning via the modified candidate elimination algorithm (Algorithm 2).

When producer receives any request from the consumer, the learning set of the producer is trained with this request as a positive sample. The learning components, the most specific set S and the most general set G are actively used in offering service. The most general set, G is used by the producer in order to avoid offering the services, which will be rejected by the consumer agent. In other words, it filters the service set from the undesired services, since G contains hypotheses that are consistent with the requests of the consumer. The most specific set, S is used in order to find best offer, which is similar to the consumer's preferences. Since the most specific set S holds the previous requests and current request, estimating similarity between this set and every service in the service list is very convenient to find the best offer from the service list. The algorithm used in offering a service to the consumer is called

Table 2: How Modified CEA works

Sample Type	Sample	The general sets	The most specific set
+	(Full, Strong, White)	{ (?, ?, ?) }	{ (Full,Strong,White) }
-	(Full, Delicate, Rose)	{ { (?-Full), ?, ? }, { ?, (?-Delicate), ? }, {?, ?, (?-Rose) }	{ (Full,Strong,White) }
+	(Medium, Moderate, Red)	{ { (?-Full), ?, ? }, { ?, (?-Delicate), ? }, {?, ?, (?-Rose) }	{ {(Full,Strong,White)}, {(Medium,Moderate,Red) } }

Similarity with Modified Version Space (SMVS) and is shown in Algorithm3.

When the consumer starts the interaction with the producer agent, producer agent loads all related services to the service list object. This list constitutes the provider's inventory of services. Upon receiving a request, if the producer can offer an exactly matching service, then it does so. For example, for a wine this corresponds to selling a wine that matches the specified features of the consumer's request identically. When the producer cannot offer the service as requested, it tries to find the service that is the most similar to the services that have been requested by the consumer during the negotiation. To do this, the producer has to compute the similarity between the services it can offer and the services that have been requested.

Algorithm 3 Offering Service Algorithm (SMVS)

Require: The learning algorithm is trained with the request as a positive example

1: **if** it is the first time **then**
2: $ServiceList \leftarrow$All related available services from service ontology {the producer agent can provide }
3: **end if**
4: **if** any s in ServiceList is equal to the request **then**
5: Offer s to the consumer
6: **else**
7: Remove each s in ServiceList which is not covered by the general sets in learning object
8: **for all** s in ServiceList **do**
9: Estimate similarity between s and the most specific set in learning object
10: Hold the maximum similarity in $MaxSim$
11: Hold the count of services owing the maximum similarity in $ServiceCount$
12: **end for**
13: **if** $ServiceCount > 1$ **then**
14: Find rating values of s whose similarity is equal to $MaxSim$
15: Offer the services whose rate value is the biggest
16: **else**
17: Offer the service whose similarity is equal to $MaxSim$
18: **end if**
19: **end if**

Similarity can be estimated with one of the similarity metrics. There are several similarity metrics used in case based reasoning system such as weighted sum of Euclidean distance, Hamming distance and other useful similarity metrics can be chosen [14]. We have used Tversky's similarity measure [17]. This metric compares two vectors in terms of the number of exactly matching features.

In Equation 1, $common$ represents the number of matched attributes whereas $different$ represents the number of the different attributes. For now, we assume α and β is equal to each other. In the future, we will try different weight values and select the best one by comparing them.

$$SM_{pq} = \frac{\alpha(common)}{\alpha(common) + \beta(different)} \quad (1)$$

This formula is for calculating the similarity between two feature vectors. In our system while the list of services that can be offered by the producer are each a feature vector, the most specific set S is not a feature vector. S consists of hypotheses of feature vectors. Therefore, we estimate the similarity of each hypothesis inside the most specific set S and then take the average of the similarities. Moreover, each hypothesis in S may involve a different number of service instances.

EXAMPLE 8. *Assume that S contains the following two hypothesis: { {Light, Moderate, (Red, White)} , {Full, Strong, Rose}}. Take service s as (Light, Strong, Rose). Then the similarity of the first one is equal to 1/3 and the second one is equal to 2/3 in accordance with Equation 1. Normally, we take the average of it and obtain (1/3+2/3)/2, equally 1/2. However, the first hypothesis involves the effect of two requests and the second hypothesis involves only one request. As a result, we expect the effect of the first hypothesis should be greater than that of the second.*

Therefore, we decide to calculate the average similarity by considering the number of samples that hypotheses cover. Let c_h denote the number of samples that hypothesis h can cover and $(SM_{(h,service)})$ denotes the similarity of hypothesis h with the given service s. We compute the similarity of each hypothesis with the given service and weight them with the number of samples they cover. We find the similarity by dividing the weighted sum of the similarities of all hypotheses in S with the service by the number of all samples that are covered in S.

$$AVG - SM_{(service,S)} = \frac{\sum_{|h|}^{|S|} (c_h * SM_{(h,service)})}{\sum_{|h|}^{|S|} c_h} \quad (2)$$

EXAMPLE 9. *For the above example, the similarity of (Light, Strong, Rose) with the specific set is (2*1/3+2/3)/3, equally 4/9. The possible number of samples that a hypothesis covers can be estimated with multiplying cardinalities of each attribute. For example, the cardinality of the first attribute is 2 and the others is equal to 1 for the given hypothesis such as {Light, Moderate, (Red, White)}. When we multiply them, we obtain two (2*1*1=2).*

Offering a service is actually similar to case-based approach. We select the feature values of the service vector in the service list obtained from the ontology and filtered by the most general set G, which is the most similar to the specific set in our learning algorithm whereas in case-based approach, the most similar one to the current the request is chosen by losing the past request information. In other words, when we investigate the similarity, we do not only look at the current request, but also look at the previous request of the consumer by the help of the most specific set covering all requests during the negotiation phase.

If there is more than one service with the maximum similarity, the producer agent chooses one of them in accordance with the rating value of it; the higher rating is more attractive for the consumer agent. After offering service, there are two options for consumer: to accept or reject. If the consumer rejects, the producer trains the learning set with the offer as a negative sample. This will continue until the consumer takes a service or until a certain time interval.

4. DEVELOPED SYSTEM

We first give a brief overview of our setup and then discuss our test cases.

4.1 Architectural Setup

We have implemented our architecture in Java. To ease testing of the system, the consumer agents has a user interface that allows us to enter various requests. The producer agent fully automates the learning and service offering operations explained before. We use OWL[13] as our ontology language and KAON2[9] as our ontology reasoner. As mentioned before, the system uses the well-known wine ontology with an extension of *WineProduct* class. This class is necessary for the producer to record the wines it sells; i.e., its inventory information.

This ontology also includes the individuals of the classes which are in the range of *hasWine* property and also involves the individuals of *WineProduct* classes. Following is an example of such individuals.

```
<Beaujolais rdf:ID="ChateauMorgon">
  <hasMaker  rdf:resource="#ChateauMorgon" />
  <madeFromGrape rdf:resource="#GamayGrape" />
  <hasSugar  rdf:resource="#Dry" />
  <hasFlavor rdf:resource="#Delicate" />
  <hasBody   rdf:resource="#Light" />
  <hasColor  rdf:resource="#Red" />
  <locatedIn rdf:resource="#BeaujolaisRegion"/>
</Beaujolais>
<WineProduct rdf:ID="WineProduct1">
  <Year>1975</Year>
  <Count>1</Count>
  <Rating>4</Rating>
  <hasWine rdf:resource= "#ChateauMorgon"/>
</WineProduct>
```

The year information has not been used to keep experiments simple. The count value keeps track of the number of products in inventory and is used to check availability of the product. The rating value is an indicator of the quality of the wine product. Among two similar wines, the one with higher rating value is preferred over the other. The price of the wine is deliberately left out to emphasize that we are concerned with the value of the service. Table 3 shows 19 individuals of WineProduct, which are used in the testing phase.

4.2 Example

Consider a customer that is planning on buying wine. For the customer, the color, location and winery of the wine are more important than the remaining features such as the body, flavor and so on. In other words, the consumer can be convinced to buy a wine product whose flavor is different from the one that the consumer initially specifies in its request, but she cannot be convinced to buy the wine whose color is different from her initial specification. Obviously, these preferences are not known by the service provider ahead of time.

Table 4 depicts the interactions between the consumer and the producer. The consumer starts the negotiation by generating the following request [Elyse, MerlotGrape, OffDry, Strong, Medium, Red, NapaRegion] where Elyse is the name of winery, MerlotGrape is the type of grape that wine is made from, OffDry indicates the sugar level, Strong is the flavor of wine where Medium is an indicator for the body of the wine, the color of wine is Red and NapaRegion is the region of the wine.

Table 5: Estimated Similarities for the Example

Product ID	After Req.1	After Req.2	After Req.3
P1	0.143	0.143	0.190
P2	0.429	**0.429**	-
P3	**0.571**	-	-
P4	0.286	0.286	0.238
P5	0.429	0.429	0.380
P6	0.286	0.286	0.286
P7	0.286	0.286	0.238
P8	0.143	0.143	0.143
P9	0.143	0.143	0.143
P10	0.143	0.143	0.095
P11	0.143	0.143	0.095
P12	0.0	0.0	0.0
P13	0.286	0.286	0.238
P14	0.143	0.286	0.333
P15	0.143	0.286	0.238
P16	0.143	0.143	0.143
P17	0.0	0.0	0.048
P18	0.429	0.429	**0.429**
P19	0.286	0.286	0.286
Max. Similarity:	**0.571**	**0.4289**	**0.429**

When this request is taken by the producer, it firstly trains its learning set, treating this request as a positive sample as explained before. Since this is the only positive example so far, it constitutes the most specific set S in accordance with the modified CEA. Then, it estimates the similarities of each available service in the service list of the producer shown in Table 3 with the most specific set. The output of the similarities estimated by the producer agent using Equation 1 is displayed in Table 5.

According to the similarities of the first time, the most similar product is selected as $P3$ having the maximum similarity value 0.571 (= 4/7 = (common / (common + different)). This service supplies the same color and location information with the initial request. However, its winery is different from the one that the consumer agent strictly wants. Therefore, the consumer rejects the offer and makes another request, which is also in accordance with the customer's preferences (i.e., only the grape type is changed). At this point, the producer inserts the rejected offer as a negative example into its learning algorithm and removes it from the service list. Following this, the producer recalculates the similarity of the services in its service list with S, which is equivalent to $\{$ (Elyse , [MerlotGrape, MalbecGrape], OffDry, Strong, Medium, Red, NapaRegion) $\}$. At this time, there are three products with the maximum similarity value 0.429.

1. Similarity of P2 = (2* 3/7) /2 =0.429
2. Similarity of P5 = (2* 3/7)/2 =0.429
3. Similarity of P18 = (2* 3/7)/2 =0.429

The producer prefers to offer the second product since its rating value is higher than the other products. However, the winery requirement of the customer is not supported by this offer. Therefore,

Table 3: Available Services

WineProduct1 :ChateauMorgonBeaujolais	Meta Data: {Year=1975, Count=1, Rating=4}
{ChateauMorgon, GamayGrape, Dry, Delicate, Light, Red, BeaujolaisRegion}	
WineProduct2 :WhitehallLaneCabernetFranc	Meta Data: {Year=1985, Count=2, Rating=5}
{WhitehallLane, CabernetFrancGrape, Dry, Moderate, Medium, Red, NapaRegion}	
WineProduct3 :FormanCabernetSauvignon	Meta Data: {Year=1985, Count=1, Rating=4 }
{ Forman, CabernetSauvignonGrape, Dry, Strong, Medium, Red, NapaRegion }	
WineProduct4 :MariettaCabernetSauvignon	Meta Data: {Year=1985, Count=1, Rating=4}
{ Marietta, CabernetSauvignonGrape, Dry, Moderate, Medium, Red, SonomaRegion }	
WineProduct5 :PageMillWineryCabernetSauvignon	Meta Data: {Year=1976, Count=3, Rating=3 }
{ PageMillWinery, CabernetSauvignonGrape, Dry, Moderate, Medium, Red, NapaRegion }	
WineProduct6 :SantaCruzMountainVineyardCabernetSauvignon	Meta Data: {Year=1989, Count=4, Rating=4 }
{ SantaCruzMountainVineyard, CabernetSauvignonGrape, Dry, Strong, Full, Red, SantaCruzMountainsRegion }	
WineProduct7 :BancroftChardonnay	Meta Data: {Year=1991, Count=1, Rating=2 }
{Bancroft, ChardonnayGrape, Dry, Moderate, Medium, White, NapaRegion }	
WineProduct8 :FormanChardonnay	Meta Data: {Year=1975, Count=2, Rating=5 }
{ Forman, ChardonnayGrape, Dry, Moderate, Full, White, NapaRegion }	
WineProduct9 :MountadamChardonnay	Meta Data: {Year=1985, Count=2, Rating=4 }
{Mountadam, ChardonnayGrape, Dry, Strong, Full, White, SouthAustraliaRegion }	
WineProduct10 :MountEdenVineyardEdnaValleyChardonnay	Meta Data: {Year=1989, Count=1, Rating=4 }
{MountEdenVineyard, ChardonnayGrape, Dry, Moderate, Medium, White, EdnaValleyRegion }	
WineProduct11 :PeterMccoyChardonnay	Meta Data: {Year=1991, Count=10, Rating=3 }
{PeterMccoy, ChardonnayGrape, Dry, Moderate, Medium, White, SonomaRegion }	
WineProduct12 :FoxenCheninBlanc	Meta Data: {Year=1981, Count=7, Rating=4 }
{Foxen, CheninBlancGrape, Dry, Moderate, Full, White, SantaBarbaraRegion }	
WineProduct13 :VentanaCheninBlanc	Meta Data: {Year=1976, Count=5, Rating=5 }
{Ventana, CheninBlancGrape, OffDry, Moderate, Medium, White, CentralCoastRegion }	
WineProduct14 :WhitehallLanePrimavera	Meta Data: {Year=1974, Count=3, Rating=5 }
{CongressSprings, MalbecGrape, Sweet, Delicate, Light, Rose, NapaRegion }	
WineProduct15 :SelaksIceWine	Meta Data: {Year=1984, Count=13, Rating=4 }
{Selaks, MalbecGrape, Sweet, Moderate, Medium, White, NewZealandRegion }	
WineProduct16 :SchlossRothermelTrochenbierenausleseRiesling	Meta Data: {Year=1987, Count=23, Rating=3 }
{SchlossRothermel, SangioveseGrape, Sweet, Strong, Full, Rose, GermanyRegion }	
WineProduct17 :StGenevieveTexasWhite	Meta Data: {Year=1991, Count=29, Rating=3 }
{StGenevieve, GamayGrape, Dry, Moderate, Light, White, CentralTexasRegion }	
WineProduct18 :ElyseZinfandel	Meta Data: {Year=1976, Count=7, Rating=4 }
{Elyse, ZinfandelGrape, Dry, Moderate, Full, Red, NapaRegion }	
WineProduct19 :ChateauMargaux	Meta Data: {Year=1981, Count=15, Rating=4 }
{ChateauMargauxWinery, MerlotGrape, Dry, Delicate, Light, Red, MargauxRegion }	

the consumer agent will reject it. The next request, the consumer changes the body attribute from strong to light. The producer recalculates the similarity with the most specific set which is equal to { {Elyse,(MerlotGrape, MalbecGrape),OffDry, Strong, Medium, Red, NapaRegion}, { Elyse, MalbecGrape,OffDry, Strong, Light,Red, NapaRegion }}. At the third time, the maximum similarity is equal to 0.429. Therefore, P18 is offered to the customer.

$$Similarity_{P18,S} = (2 * (3/7) + 3/7)/3 = 0.429 \qquad (3)$$

Since this offer is consistent with the customer's preferences, the consumer agent accepts the offer. When we investigate the offers, we can see that the producer agent learns that the winery is an important feature for the customer. In the third iteration, the producer finds a wine product that matches the consumer's preferences.

5. PERFORMANCE EVALUATION

After giving a detailed example, we evaluate the performance of the proposed system (*SMVS*) by comparing it with the two other approaches that do not use inductive learning. The first compared system randomly generates counter offers by picking random products for the user. We call this approach *Random Offering*. The second system we use (*SCR*) offers the most similar service to the current request by ignoring the prior requests made.

We use five test cases for this evaluation process. Each test case contains a list of preferences for the user and number of matches from the product list. Also, the performance depends on the initial request, thus, we repeat our experiments with different initial requests.

Consequently, for each case, we run the algorithms five times with several variations of the requests. In each experiment, we count the number of iterations that need to be made to reach an agreement. We take the average of the this number in order to evaluate these systems fairly. As is customary, we start each test case with the same initial request.

5.1 Scenario 1

The customer wants to buy any wine whose sweetness degree is dry. There are 15 products in the inventory that meets this condition. Table 6 show the average of five runs for all three approaches.

Table 4: An Example Negotiation

REQUEST - 1	[Elyse , MerlotGrape, OffDry, Strong, Medium, Red, NapaRegion]
OFFER - 1	[Forman, CabernetSauvignonGrape, Dry, Strong, Medium, Red, NapaRegion]
REQUEST - 2	[Elyse, MalbecGrape, OffDry, Strong, Medium, Red, NapaRegion]
OFFER - 2	[WhitehallLane, CabernetFrancGrape, Dry, Moderate, Medium, Red, NapaRegion]
REQUEST - 3	[Elyse, MalbecGrape, OffDry, Strong, Light, Red, NapaRegion]
OFFER - 3	[Elyse, ZinfandelGrape, Dry, Moderate, Full, Red, NapaRegion]

Here, *number of iterations* represents the number of steps it takes for the customer's request to be fulfilled by the producer.

SMVS score is slightly better than the SCR Technique. During the test, we see that both SMVS and SCR techniques offer the same service at the beginning of the negotiation phase since the most specific set of the modified version space is equal to the initial request at the first time. However, in the following interaction their behaviors change. Furthermore, for this scenario, Random Offering is as good as SMVS since the inventory involves a large number of available services that are consistent with customer's preferences and the probability of offering a convenient service is very high. This probability is equal to 15/19 at the beginning of the negotiation and increases over time.

Table 6: Number of iterations for Scenario 1

	SMVS	SCR	Random Offering
R1	2	3	1
R2	1	1	1
R3	1	1	2
R4	1	1	1
R5	1	1	1
Average:	**1.2**	**1.4**	**1.2**

In order to see the performance differences between the SMVS and SCR in detail, we give a comparative run-time example. Table 7 indicates the request-offer pairs of SMVS whereas Table 8 shows those of the SCR Technique.

The consumer starts with the request [Ventana, MalbecGrape, Dry, Delicate, Light, Rose, CotesDOrRegion]. The similarities estimated for both SMVS and SCR is shown in Table 9. It is seen that the similarity values are the same for both system after initial request. The maximum similarity is calculated as 4/7, equally 0.5714. As a result, both systems offer the product P14. After second request, maximum similarity is estimated for SCR as 2/7, equally 0.2857. This similarity is based on only current request. Then, the product P13 is offered by SCR system but not accepted by the consumer agent. On the other hand, the similarity calculation for SMVS is based on the most specific set, which is equal to {Ventana, MalbecGrape, Dry, Delicate,(Light, Medium), Rose, CotesDOrRegion}. According to this calculation, the similarity of product P19 is equal to 3/7, whereas the similarity of product P13 is 2/7. Consequently, SMVS offers product P19 and it will be accepted by the consumer agent. The third offer will be product P3 for SCR technique. In this example, it is obvious that considering the most specific set involving all previous requests in the interaction is beneficial for offering closer services to the customer's preferences.

5.2 Scenario 2

The customer wants to buy any wine, which is red and dry. There are eight products (out of 19) in the stock that meets this condition. When we look at the results of SMVS, SCR and Random Offering, the performance of SMVS and SCR is equal and better than that of Random Offering (Table 10). At the beginning of the negotiation, the probability of offering an acceptable service for Random Offering is equal to 8/19. Therefore, the number of iterations is increased with respect to the first case.

Table 10: Number of iterations for Scenario 2

	SMVS	SCR	Random Offering
R1	2	2	3
R2	2	2	1
R3	1	1	1
R4	1	1	3
R5	1	1	5
Average:	**1.4**	**1.4**	**2.6**

5.3 Scenario 3

The customer wants to buy any wine, which is red, dry and moderate. There are four products meeting this condition. The test results in Table 11 indicates that SMVS shows the best performance and the worst performance belongs to Random Offering. When the number of available service that match the customer's preferences decrease, the performance difference between the SMVS and other techniques becomes clear.

Table 11: Number of iterations for Scenario 3

	SMVS	SCR	Random Offering
R1	2	2	3
R2	1	1	10
R3	1	1	4
R4	1	1	2
R5	2	4	3
Average:	**1.4**	**1.8**	**4.4**

5.4 Scenario 4

The customer wants to buy any wine, which is strong and red. There are only two products in the inventory that meets this condition. Again, we look at the average of five runs for three approaches. Table 12 shows that SMVS approximately finds the suitable offer at 2.2 iterations whereas SCR finds the convenient counteroffer at 2.8 iterations. The performance of the Random Offering is comparatively low.

5.5 Scenario 5

The customer wants to buy any wine whose flavor is strong and color is red or rose. There are three products meeting this condition. This case is good to see the effect of learning the disjunctive concept such as (Color = Red or Rose) with the conjunctives (and Flavor= Strong). In some situation, SCR technique can find the convenient service sooner than SMVS and in some situation SMVS

Table 7: Running of Scenario 1 using SMVS

REQUEST - 1	[Ventana, MalbecGrape, Dry, Delicate, Light, Rose, CotesDOrRegion]
OFFER - 1	P14 = [CongressSprings, MalbecGrape, Sweet, Delicate, Light, Rose, NapaRegion]
REQUEST - 2	[Ventana, MalbecGrape, Dry, Delicate, Medium, Rose, CotesDOrRegion]
OFFER - 2	P19 = [ChateauMargauxWinery, MerlotGrape, Dry, Delicate, Light, Red, MargauxRegion]

Table 8: Running of Scenario 1 using SCR

REQUEST - 1	[Ventana, MalbecGrape, Dry, Delicate, Light, Rose, CotesDOrRegion]
OFFER - 1	P14 = [CongressSprings, MalbecGrape, Sweet, Delicate, Light, Rose, NapaRegion]
REQUEST - 2	[Ventana, MalbecGrape, Dry, Delicate, Medium, Rose, CotesDOrRegion]
OFFER - 2	P13 = [Ventana, CheninBlancGrape, OffDry, Moderate, Medium, White, CentralCoastRegion]
REQUEST - 3	[Ventana, MalbecGrape, Dry, Strong, Medium, Rose, CotesDOrRegion]
OFFER - 3	P3 = [Forman, CabernetSauvignonGrape, Dry, Strong, Medium, Red, NapaRegion]

Table 12: Number of iterations for Scenario 4

	SMVS	SCR	Random Offering
R1	2	4	16
R2	4	4	6
R3	1	1	11
R4	1	1	10
R5	3	4	5
Average:	**2.2**	**2.8**	**9.6**

can show better performance. This depends on the requests. Average performance of SMVS over five runs is better than that of SCR as seen in Table 13.

Table 13: Number of iterations for Scenario 5

	SMVS	SCR	Random Offering
R1	2	6	7
R2	1	1	10
R3	3	2	13
R4	3	3	6
R5	1	1	2
Average:	**2**	**2.6**	**7.6**

5.6 Comparison over Scenarios

Table 14 compares the three approaches over all scenarios. The results indicate that SMVS performance is higher than both of the other systems. When the number of possible services is decreasing, the time interval for offering an acceptable service gets longer in Random Offering. When the large parts of inventory is compatible with the customer's preferences as in the first test case, the performance of all techniques are nearly same.

Table 14: Average Number of Iterations for Five Scenarios

	SMVS	SCR	Random Offering
Scenario 1:	1.2	1.4	1.2
Scenario 2:	1.4	1.4	2.3
Scenario 3:	1.4	1.8	3.7
Scenario 4:	2.2	2.8	7.8
Scenario 5:	2	2.6	6.3
Average of all cases:	**1.64**	**2**	**4.26**

6. DISCUSSION

We review the recent literature in comparison to our work. Debenham [5] studies the management of the single-issue and multi-issue negotiation processes using the market data and information over the Internet. Each transaction such as request and offer is considered as a business process and have some constraints such as time. A negotiation should be completed up to a certain time. Brzostowski and Kowalczyk propose an approach in order to select an appropriate negotiation partner by investigating previous multi-attribute negotiations [2]. For achieving this, they use case-based reasoning. They emphasize the positive effect of finding the right partner. Their approach is probabilistic since the behavior of the partners can be changed by the next time. In our approach, we are interested in negotiation the content of the service. After the consumer and producer agree on the service, price-oriented negotiation mechanisms can be used to agree on the price.

Tama *et al.* [16] propose a new approach based on ontology for negotiation. According to their approach, the negotiation protocols used in e-commerce can be modeled as ontologies. Thus, the agents can perform negotiation protocol by using this shared ontology without the need of being hard coded of negotiation protocol details. This shared ontology provides the vocabulary of the negotiation process for the agents, which can behave dynamically. However, Tama *et al.* do not provide a method for learning preferences during negotiation as we have done here.

Ontology can be used in service discovery for better results than that based on syntactic matching. Broens *et al.* [1] propose a context-aware, ontology-based service discovery. Matching capabilities can be increased by using the semantic information. For instance, when the user requests for selling music, selling CD will be valid since the CD is a subtype of music. They classify the services according to their properties and model the service and service types in an ontology. To find a service, they consider contextual information in addition to inputs and outputs.

Faratin *et al.* propose a multi-issue negotiation mechanism based on trade-offs [6]. The service variables for the negotiation process such as price, quality of the service, delivery time and so on are considered traded-offs against each other (i.e., higher price for earlier delivery). They generate a heuristic model for trade-offs including fuzzy similarity estimation and a hill-climbing exploration for possibly acceptable offers. Although we address a similar problem, our main target is to learn the preference of the customer by the help of inductive learning and generate counter-offers in accordance with these learned preferences. Faratin *et al.* only use the last offer made by the consumer in calculating the similarity for choosing counter offer. Unlike them, we also take into account the

Table 9: The Estimated Similarities for Scenario 1

	Similarities for SMVS		Similarities for SCR		
Product ID	After Request-1	After Request-2	After Request-1	After Request-2	After Request-3
P1	0.4286	0.4286	0.4286	0.2857	0.1429
P2	0.1429	0.2857	0.1429	0.2857	0.2857
P3	0.1429	0.2857	0.1429	0.2857	**0.4286**
P4	0.1429	0.2857	0.1429	0.2857	0.2857
P5	0.1429	0.2857	0.1429	0.2857	0.2857
P6	0.1429	0.1429	0.1429	0.1429	0.2857
P7	0.1429	0.2857	0.1429	0.2857	0.2857
P8	0.1429	0.1429	0.1429	0.1429	0.1429
P9	0.1429	0.1429	0.1429	0.14295	0.2857
P10	0.1429	0.2857	0.1429	0.2857	0.2857
P11	0.1429	0.2857	0.1429	0.2857	0.2857
P12	0.1429	0.1429	0.1429	0.1429	0.1429
P13	0.1429	0.2857	0.1429	**0.2857**	-
P14	**0.5714**	-	**0.5714**	-	-
P15	0.1429	0.2857	0.1429	0.2857	0.2857
P16	0.1429	0.1429	0.1429	0.1429	0.2857
P17	0.2857	0.2857	0.2857	0.1429	0.1429
P18	0.1429	0.1429	0.1429	0.1429	0.1429
P19	0.4286	**0.4286**	0.4286	0.2857	0.1429
Maximum Similarity:	**0.5714**	**0.4286**	**0.5714**	**0.2857**	**0.4286**

previous requests of the consumer using Modified Version Space. In their experiments, Faratin *et al.* assume that the weights for service variables are fixed a priori. On the contrary, we aim to learn these preferences through inductive learning.

Our approach is open for improvements. First, the preferences of the consumer agent may change during the negotiation. It would be interesting to extend the learning algorithm to deal with dynamic changes in the requests of the consumer agent. The preferences can be enriched with subtle relations on price. Second, semantic closeness among the feature values can be used in the estimation of the similarity of the services with the most specific set in the learning phase in order to obtain better results. Up to now, when we compare two attributes, we assign 0 for dissimilarity and 1 for similarity. However, we can use intermediate values to express the closeness of the attributes. For example, 0.5 can be used for red and rose whereas 0.2 can be used for white and rose. Third, currently, the producer only tries to learn the consumer's demand without considering its own preferences. Incorporating the business strategies of the producer will allow more realistic scenarios to be tested. These are interesting directions that we will pursue in our future work.

7. REFERENCES

[1] T. Broens, S. Pokraev, M. van Sinderen, J. Koolwaaij, and P. D. Costa. Context-aware, ontology-based service discovery. In *EUSAI*, volume 3295 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2004.

[2] J. Brzostowski and R. Kowalczyk. On possibilistic case-based reasoning for selecting partners for multi-attribute agent negotiation. In *Proc. of the 4th Intl. Joint Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 273–278, July 2005.

[3] L. Busch and I. Horstman. A comment on issue-by-issue negotiations. *Games and Economic Beh.*, 19:144–148, 1997.

[4] M. C. Daconta, L. J. Obrst, and K. T. Smith. *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*. Wiley, Indianapolis, 2003.

[5] J. K. Debenham. Managing e-market negotiation in context with a multiagent system. In *Proceedings Twenty First International Conference on Knowledge Based Systems and Applied Artificial Intelligence, ES'2002: Applications and Innovations in Expert Systems X*, Cambridge, UK, 2002.

[6] P. Faratin, C. Sierra, and N. R. Jennings. Using similarity criteria to make issue trade-offs in automated negotiations. *Artificial Intelligence*, 142(2):205–237, 2002.

[7] S. Fatima, M. Wooldridge, and N. Jennings. Optimal agents for multi-issue negotiation. In *Proceeding of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 129–136, Melbourne, Australia, July 2003. ACM Press.

[8] D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster. *Spinning the Semantic Web*. MIT Press, Cambridge, 2003.

[9] KAON2, 2003. http://kaon2.semanticweb.org.

[10] P. Maes, R. H. Guttman, and A. G. Moukas. Agents that buy and sell. *Communications of the ACM*, 42(3):81–91, 1999.

[11] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[12] T. M. Mitchell. *Machine Learning*. McGraw Hill, NY, 1997.

[13] OWL. Owl web ontology language guide, 2003. http://www.w3.org/TR/2003/CR-owl-guide-20030818/.

[14] S. K. Pal and S. C. K. Shiu. *Foundations of Soft Case-Based Reasoning*. John Wiley & Sons, New Jersey, 2004.

[15] M. P. Singh. Value-oriented electronic commerce. *IEEE Internet Computing*, 3(3):6–7, 1999.

[16] V. Tamma, S. Phelps, I. Dickinson, and M. Wooldridge. Ontologies for supporting negotiation in e-commerce. *Engineering Applications of Artificial Intelligence*, 18:223–236, 2005.

[17] A. Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.

[18] Wine, 2003. http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine.rdf.