The Sales Agent

An online Product Recommendation System

Purshottam Vishwakarma
M.S. Computer Science
Indiana University
pvishwak@indiana.edu

Bitan Saha
M.S. Computer Science
Indiana University
pvishwak@indiana.edu

***Abstract*— We have stepped into an era where innumerable brands with varied specifications are available for any product in the market especially the electronic products like laptops, mobile phones, camera, DVD player, etc. A buyer is left inevitably in a state of muddiness as to which brand to buy and with what specifications. We intend to solve this problem faced by a common man by developing a system which will use the existing knowledge base to recommend the right product to a given customer as per his needs. The system can be used by anyone who may or may not have the complete knowledge of the product domain. We have created an online system called "The Sales Agent" which would help the buyer to choose the best product. This system also learns from the feedback mechanism and updates its rules to in order to retrieve the best recommendation from the plethora of products available to the user. We implemented this system using knowledge-based AI methods case based reasoning and rule based reasoning. We also describe in detail the algorithm used to rank the search results according to the product attributes provided by the user which is the crux of this system.**

***Keywords*- Knowledge Base, Case based reasoning, Rule based resoning, Artificial Intelligence.**

I. INTRODUCTION

Every person has experienced the confusion in mind with respect to which product he/she should buy given the fact that there are multitude of options available in the market place. This brings a whole new opportunity to help buyers in choosing the best product and attaining higher self-satisfaction. It is not every time possible for us to go to a store and ask the sales representative to help the purchase. Also it is possible that the recommendation of the sales person could be biased to a particular brand. For these reasons we feel motivated to accept this challenging problem and help customers with the unbiased best product selection. The solution to this problem can best be implemented using knowledge-based AI methods such as *Case Based Reasoning* and *Rule Based Reasoning*.

II. GOAL

The goal of the Sales Agent is to recommend the most suitable product(s) to a customer as per his/her needs and reason the recommendation.

This system will eliminate human interference while evaluating the right product for a customer and increase customer satisfaction.

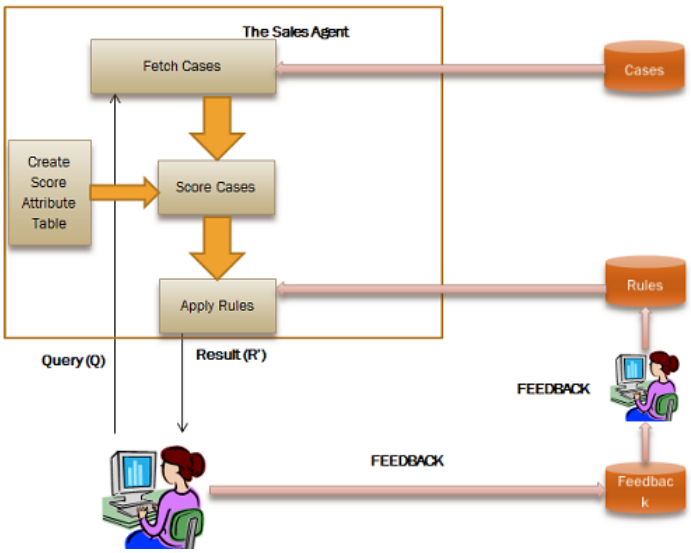


Figure 1. The Sales Agent Architecture

III. ARCHITECTURE

The Sales Agent implements a master/slave model. The user is provided with a web interface to provide the attributes which it feels to be important and along with its level of importance. The sales Agent engine thereby fetches the relevant cases from the knowledge base based on the relevant attributes specified by the user. The sales agent engine then scores each of the cases using the score attribute table. The rules are then applied on the filtered cases to give back the results to the user. Also there is a provision of feedback mechanism. The feedback

from the users is entered in the rules table which is applied by the sales agent engine.

We explain the internal working of the Sales Agent in a step by step process. Step 1 comprises of receiving the attribute request from the user as shown in figure 1. This is received by the Fetch cases module of the Sales Agent.

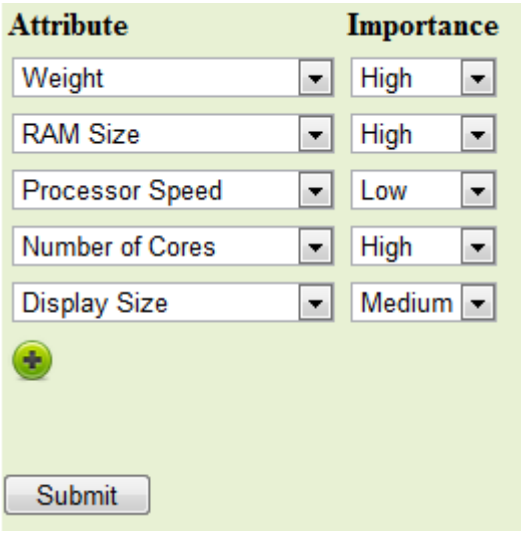


Figure2: The web interface for entering the attributes

A. *Creation of Attribute Score Table*

This is step 2 of the entire process. As per the user requirement attributes score table is determined which assigns a weight to each attribute.

The *score distribution scope* is fixed to 1000 i.e. we are going to score each case out of 1000. As we have total 15 attributes, each attribute gets 66.67(1000/15) points initially. Let's say the user selects 5 attributes out of 15. The user would also have selected the importance of the attributes in the range of high (30), medium (20) or low (10).

$$\%Contribution = \frac{Attribute\ Weight}{Sum\ of\ all\ weights}\ X\ 100$$

We determine the percentage contribution of each selected attribute as given below:

Attributes	Weight	RAM	Processor Speed	No. of Cores	Display Size
Importance	High	High	Low	High	Medium
Imp. Value	30	30	10	30	20
%Contribution	25.00	25.00	8.33	25.00	16.67

$$NonSelected\ attribute\ contribution = 1\% \frac{Total\ distribution\ scope}{Total\ no.\ of\ Attributes}\ X\ No.\ of\ Nonselected\ attributes$$

$$= 1\% \frac{1000}{15}\ X\ 10$$

$$= 6.67$$

In this example, Total Non-selected attributes = 10

This way we come up with the Score Attribute table as shown in table 1. The selected attributes are highlighted in Red.

At the end of step 2, we have *weights* assigned to each attribute.

Score Attribute Table		
Attributes	**Score**	
Brand	60.00	66.67 – 6.67
Display Size	83.33	66.67 + 16.67
Processor Speed	75.00	66.67 + 8.33
No of Cores	91.67	66.67 + 25
Processor Brand	60.00	66.67 – 6.67
RAM Size	91.67	66.67 + 25
HDD Size	60.00	66.67 – 6.67
Price	60.00	66.67 – 6.67
Battery Life	60.00	66.67 – 6.67
Cache Size	60.00	66.67 – 6.67
Graphics Card	60.00	66.67 – 6.67
OS	60.00	66.67 – 6.67
Weight	91.67	66.67 + 25
Warranty	60.00	66.67 – 6.67
No of USB Ports	60.00	66.67 – 6.67

Table 1. Score Attribute Table

B. *Case Retrieval*

This is step 3 of the process. Fetch the relevant cases from the knowledge base depending on the attributes along with its importance level selected by the user. Reason to have such a huge selected case base is to obtain varied score densities). This part of the process uses ***Case Based Reasoning*** approach.

The selection process can be explained with the example taken in Step A.

After determining all the available display sizes (10, 11, 12, 13, 14, 15, 16, and 17) we consider cases with display size 15, 16 and 17 as **high** according to below formula.

Display Sizes	10	11	12	13	14	15	16	17
Sequence Nos.	1	2	3	4	5	6	7	8

High Attribute > **(Median + High) / 2**
Medium Attribute > **Median**
Low Attribute > **Median / 2**

In this case, Median is the attribute value with sequence no. 4. High is the sequence no. 8 and low is the sequence no. 1. Therefore we consider all the case with attribute value greater than (median + high) / 2 = (4+8)/2 = 6. Hence selected cases are with attributes within the range of 6 to 8.

In the next step we score each case using the score table created as in table 1 in the steps given below:

- Get the unique values for each attribute.
- Assign sequence nos. to each attribute value.
- Divide the score of the attribute received from step 2 by the number of unique attribute values. E.g. If there are 8 different unique values of display size and the score of display size is 60 (obtained from step 2) then, 60 / 8 = 7.5 which is the weight that cumulatively gets added to the increasing attribute values.
- Multiply the sequence number of the attribute value from a fetched case with the weight to get the value of that particular attribute in that case.
- Add all the values for all the attributes to derive the unique value of the case.

In our example, we had chosen display size. We fetch all the available display sizes and assign sequence nos.

Display Sizes	10	11	12	13	14	15	16	17
Sequence Nos.	1	2	3	4	5	6	7	8
Value	7.5	15	22.5	30	37.5	45	52.5	60

The score available step 2 = 60. Therefore the maximum available score for this attribute will be 60. Now, we have retrieved cases which consist of display sizes 15, 16 and 17 from step 3. Attribute value 15 will get the score (60/8)*6 = 45. Similarly, attribute value 16 and 17 will get the scores 52.5 and 60.

Case 1: 45 + Scores from the remaining attributes obtained in a similar fashion = Total Value 1
Case 2: 52.5 + ….. = Total Value 2
Case 3: 60 + ….. = Total Value 3
…
Case n: A1 score + A2 Score + …. = Total Value n

The score table would be created for both Selected as well as non-selected attributes.
Thus at the end of step 4 we have scored each case with the help of its individual attribute scores.

Consider all the cases around and within a threshold circle from the best cases. If there are no cases within the threshold circle then consider top 5 cases. The threshold will be the average score of all the cases. Further reduction in the number cases can be done using averaging and blocking which means further dividing the entire range of scores into blocks. Thereafter we determine the number of cases in each block as *density*. This can be best explained with a concrete example.
Suppose we have **50 cases** pulled from **Step 3**. All the 50 cases will have a score assigned to it in **step 4**. We find the mean of all the scores (say the mean score is *600*. Next, we divide scores into different score ranges and determine the density (no. of cases) in each score range. The complete representation can be provided diagrammatically as shown below.

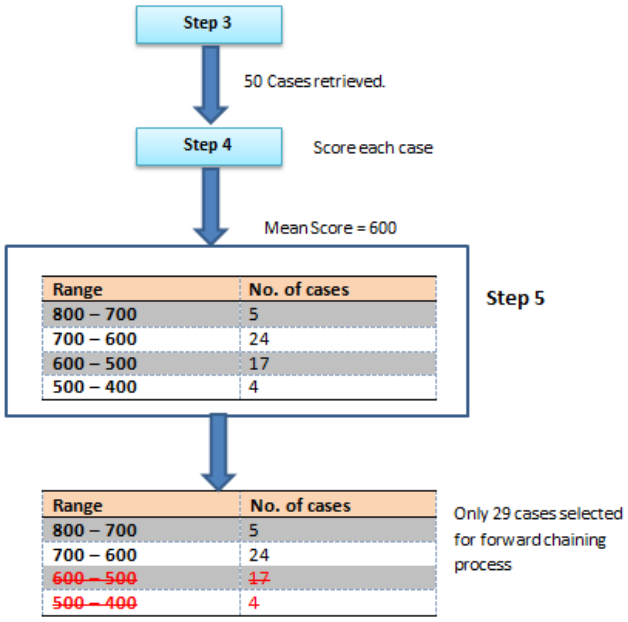


Figure 3. The flow of results from step 3 to step 5

The idea behind dividing cases into different score ranges is to obtain cases which are closely related and our algorithm will be able to distinguish such a group of cases from the rest.

Step 6: Fetch all the rules and apply it on the selected cases. The rule-based reasoning process will further eliminate cases from the previous set of cases. This includes the elimination criteria of weaker cases as selection reasoning of the strongest

case. For example, the forward chaining rules are of the form as shown below:

	Non-Selected Attribute	Selected Attribute	+/- Impact	Fact
1	No. of Cores High	Price	Negative	More number of cores increases the Price.
2	Lower the display size	Battery life	Positive	Lower display size increases the battery life

The 1st rule in the above table means as the no. of cores increase the price also increases which leads to the reduction in the score of Price which is a negative impact. Similarly, the second rule means that as the display size decreases, the battery life increases which is a positive impact.

The score of the Non-Selected Attribute determines the level of impact on the score of Selected Attribute. The function to determine the level of impact is as shown below:

Non-Selected Attribute => 1 % of its Score = x
Selected Attribute Score => Original Score ± x

We arrived on the 1% figure empirically.

Step 7: Top 3 results are returned to the user along with the reasoning obtained from the selection criteria in the previous step.

As we apply the rules which lead to the increment or decrement of the score, the facts are recorded for the reasoning process.

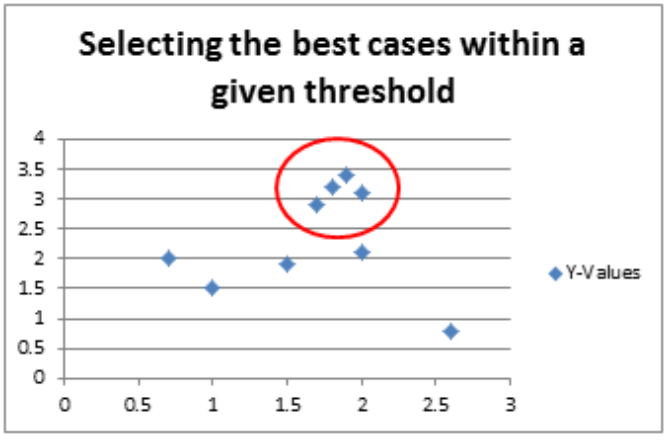


Figure 4: Selecting the best cases within a given threshold

IV. IMPLEMENTATION

We have implemented this system using Java as our programming language along with Apache Tomcat as our web server and SQL Server as database.

Database

We have created 3 tables to implement this system.

i. Product Table: This table hosts all the attributes of the product including the product type. The system is scalable to support other product types too.
ii. Rules Table: This table hosts all the rules as we discussed earlier.
iii. Product Range Table: It stores the range values of each attribute. For example, Range value for Battery life with High importance is 6. This implies we would consider all the cases whose battery life is more than 6 hours. Similarly, the range value for battery life with medium importance is 5 i.e. we would consider all the cases whose battery life is more than 5 hours.

The Sales Agent System

The Sales agent system is implemented using Java technology (jsp and servlets) which we describe in detail.

The request is received from the user in the form of HTTP request i.e. user enters the attributes of interest in the HTML form. We use Javascript to add more functionality in the html form.

ControllerServlet is a Servlet implementation class. ControllerServlet receives the HTTP request from the user, extracts all the attributes from the HTTP request and executes the business logic.

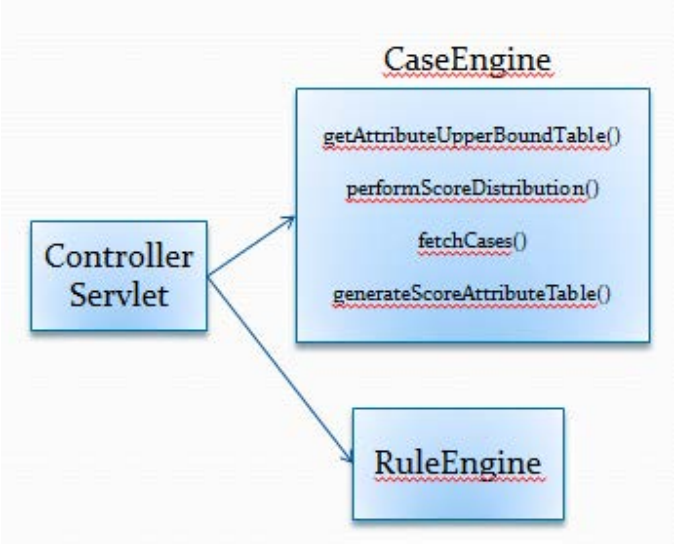


Figure 5: The control flow of the Sales Agent.

We implement the CaseEngine class to process the business logic. The following tasks are performed by the CaseEngine.

i. Generate the Initial Score Attribute Table: It gets all the attribute names from the Product table and initializes initial value of all the attributes to 1000/Total No. of Attributes. This task is performed by the getAttributeUpperBoundTable function of CaseEngine.
Below is the code snippet of the same.

```
List<String> attributeNames = DataAccess.getAllAttributesName();

// Set the Total Number Of Attributes
totalNoOfAttributes = attributeNames.size();

// Calculate the initial Average Score Of Each Attribute
initEachAttrScore = totalAttributeScope/totalNoOfAttributes;

// Populate the Table
Iterator<String> attributeNamesIterator = attributeNames.iterator();
while(attributeNamesIterator.hasNext())
{
  table.put(attributeNamesIterator.next(), ((Integer)initEachAttrScore).doubleValue());
}
```

ii. The CaseEngine then performs the score distribution i.e. it obtains the definitive scores from Non-selected attributes and distributes them among selected attributes. *performScoreDistribution* function of the CaseEngine performs the score distribution of all the selected as well as non-selected attributes. This is the Step 2 of our aforementioned process.

```
// Obtain the Contribution of Each Non Selected Attribute
eachNonSelectedAttrContibution = ( ((Double[])table.values().toArray(new Double [table.values().size()]))[0] / 100 ) * nonSelectedAttributeContributionPercentage;

// Obtain the Contribution of Non Selected Attributes Altogether
totalNonSelectedAttrContibution = eachNonSelectedAttrContibution * totalNoOfNonSelectedAttributes;

// Reduce Non-Selected Attributes Scores
List<String> selectedAttributeList = new ArrayList<String>();
Iterator<AttrImpPair> selectedAttrImpPairsIterator = selectedAttrImpPairsList.iterator();
Iterator<String> tableKeySetIte = table.keySet().iterator();

while(selectedAttrImpPairsIterator.hasNext())
{
        selectedAttributeList.add(selectedAttrImpPairsIterator.next().getAttrName());
}

while(tableKeySetIte.hasNext())
{
        String eachAttribute = tableKeySetIte.next();
        if(selectedAttributeList.contains(eachAttribute))
                continue;
        else
                table.put(eachAttribute, table.get(eachAttribute) - eachNonSelectedAttrContibution);
}

// Contribute towards Selected Attributes
double sumOfPW = 0;
HashMap<String, Double> localPercWaitageMap = new HashMap<String, Double>();
selectedAttrImpPairsIterator = selectedAttrImpPairsList.iterator();
tableKeySetIte = table.keySet().iterator();

while(selectedAttrImpPairsIterator.hasNext())
{
        sumOfPW += perctWaitageMap.get(selectedAttrImpPairsIterator.next().getImp());
}

Iterator<String> perctWaitageKeyIte = perctWaitageMap.keySet().iterator();
while(perctWaitageKeyIte.hasNext())
{
        String type = perctWaitageKeyIte.next();
        localPercWaitageMap.put( type, ((Double)( (100.00 / sumOfPW) *  perctWaitageMap.get(type))).doubleValue() );
}

while(tableKeySetIte.hasNext())
{
        String eachAttribute = tableKeySetIte.next();
        double initialValue = table.get(eachAttribute);
        String importanceLevel = null;
        if( (importanceLevel = getImportance(eachAttribute, selectedAttrImpPairsList)) == null)
                continue;
        double perctIncrement = localPercWaitageMap.get(importanceLevel);
        double increment = (totalNonSelectedAttrContibution / 100) * perctIncrement;

        if(selectedAttributeList.contains(eachAttribute))
                table.put(eachAttribute, initialValue + increment );
}
```

iii. In step 3, we fetch the relevant cases from the knowledge base based on the selected attributes and their importance level. This is performed by the *fetchCases* function of CaseEngine.

```
while(attrImpPairsListIterator.hasNext())
{
        attrRangePairList.add(DataAccess.getRange(attrImpPairsListIterator.next()));
}
// Construct The Query
Iterator<AttrRangePair> attrRangePairListIte = attrRangePairList.iterator();
while(attrRangePairListIte.hasNext())
{
        AttrRangePair eachAttrRangePair = attrRangePairListIte.next();
        fetchQuery += eachAttrRangePair.getAttrName()+" >= "+eachAttrRangePair.getRange();

        if(attrRangePairListIte.hasNext())
                fetchQuery += " AND ";
}
return DataAccess.getResults(fetchQuery);
```

iv. Now we generate the score attribute table. This is done by generateScoreAttributeTable function of CaseEngine. This is the step 4 of our process.

```
while(attributeNameSetIte.hasNext())
{
        String eachAttribute = attributeNameSetIte.next();
```

```
        TreeMap<Float, Double> uniqueScoreMap = DataAccess.getUniqueAttributeValues(eachAttribute);
        Iterator<Float> uniqueScoreMapKeySetIterator = uniqueScoreMap.keySet().iterator();

        int index = 0;
        while(uniqueScoreMapKeySetIterator.hasNext())
        {
        uniqueScoreMap.put(uniqueScoreMapKeySetIterator.next(),
(attrUpperBoundTable.get(eachAttribute)/uniqueScoreMap.keySet().size())*(index+1));
                index++;
        }
        attrScoreTable.put(eachAttribute, uniqueScoreMap);
}
```

v. After the generation of the score attribute table we assign scores to all the cases that were retrieved from step 3. This is done by scoreCases function of the CaseEngine. The results obtained after scoring the cases are written into a file on the server side for analysis purpose to check how the system is performing.

```
while(casesIterator.hasNext())
{
double score = 0;
Product eachProduct = casesIterator.next();

Iterator<String> computationalAttributesIterator = attrScoreTable.keySet().iterator();
while(computationalAttributesIterator.hasNext())
{
        String eachCompAttr = computationalAttributesIterator.next();
        score += attrScoreTable.get(eachCompAttr).get(eachProduct.getValue(eachCompAttr));
}

eachProduct.setScore(score);
}
```

vi. Now we invoke the RuleEngine to apply all the rules to the results obtained from step 5. This gives us the reasoning for all our retrieved cases.

```
while(attrImpPairsListIterator.hasNext())
{
selectedAttributes.add(attrImpPairsListIterator.next().getAttrName());
}

nonSelectedAttributes.removeAll(selectedAttributes);

Iterator<Product> casesIterator = cases.iterator();
while(casesIterator.hasNext())
{
// The Rule Engine is to run on each selected Product/Case
Product eachProduct = casesIterator.next();

// Iterating through all the Non-Selected Attributes.
Iterator<String> nonSelectedAttributeIterator = nonSelectedAttributes.iterator();

while(nonSelectedAttributeIterator.hasNext())
{
String eachNonSelectedAttribute = nonSelectedAttributeIterator.next();

// Iterate through all the Selected Attributes for every Non-Selected attribute and try to find a MATCH.
attrImpPairsListIterator = attrImpPairsList.iterator();

selectedAttributeIteration:
while(attrImpPairsListIterator.hasNext())
{
AttrImpPair eachAttrImpPair = attrImpPairsListIterator.next();
Rule rule;

if( (rule = DataAccess.getRule(eachNonSelectedAttribute, DataAccess.getAttributeRange(eachNonSelectedAttribute,
eachProduct
                .getValue(eachNonSelectedAttribute)), eachAttrImpPair)) == null )
        continue selectedAttributeIteration;

if(rule.getEffect() == Rule.EFFECT.POSITIVE)
{
        eachProduct.insertStrongFeature(rule.getRemarks());
        eachProduct.setScore( eachProduct.getScore() + (attrScoreTable.get(eachAttrImpPair.getAttrName())

        .get(eachProduct.getValue(eachAttrImpPair.getAttrName()))/100) );

}else if(rule.getEffect() == Rule.EFFECT.NEGETIVE)
{
        eachProduct.insertWeakFeature(rule.getRemarks());
        eachProduct.setScore( eachProduct.getScore() - (attrScoreTable.get(eachAttrImpPair.getAttrName())

        .get(eachProduct.getValue(eachAttrImpPair.getAttrName()))/100) );
   }
  }
 }
}
```

vii. Next we sort all the cases and print the results in the log file at the server side. The results contain the strong features and the weak features for each of the case. These results are then returned back to the user.

V. RELATED WORK

We could not find a system which would apply the rules as we mentioned before in order to filter out the cases. The closest system is Amazon which requires the user to filter the search results themselves but doesn't rank the results based on the user feedback mechanism. Our system would consider the rules i.e. positive and negative impact on the selected attributes the user is looking forward to and accordingly filter out the cases for the user.

The snapshot of the results page is shown in *appendix A and appendix B.*

VI. EVALUATION

We propose two methods to evaluate our Sales Agent results out of which only one method could actually be implemented. The other method of evaluation required the user to actually buy the product according to our recommendation and later provide the feedback after a particular duration of usage of the product. The two methods we suggested for the evaluation purposes are:

i. We will ask the customer to provide feedback on the recommendation after a period of time. This information/feedback will help us measure the performance of our system. This method of evaluation will take time as the customer will provide its feedback after he/she has used the product for a specific time period.
ii. We can ask a product domain expert to judge the system's behavior. This method of evaluation gives us instant evaluation results.

We used only method ii for the purpose of evaluation. In this method we asked 20 users to use our system and provide feedback on the results provided by the Sales Agent after they have used for some time. All these users were graduate students in computer science and hence we can consider them as experts and can count on their feedback. They were asked to rate the performance of Sales Agent from 1 to 5, 5 being the highest. 70% of the people were satisfied with the results, 30% of them rated unsatisfactory as they did not see the expected results. This was probably due to the fact that our knowledge base was limited.

VII. LIMITATIONS

The system was likely to give no results if the particular combination of attributes did not find any matching cases in the knowledge base. We intend to overcome this limitation by stopping the rule mining and the process of fetching of cases to a stage where we have results to show. If application of a particular step doesn't find any results in the knowledge base we would fall back to the results shown by the previous step. But this would deteriorate the quality of results. There is a tradeoff between showing incomplete results and showing no results that we would certainly want to consider in the future work.

VIII. CHALLENGES AND LEARNING

Applying the **rule based reasoning process**, case based representation and **scoring each matching case** has been the challenging part of this project. This project has enabled us to implement the knowledge that we garnered during the course "Knowledge based Artificial Intelligence". Implementing the knowledge base and the rule based reasoning has given us more clear understanding of the knowledge based systems. Without this project the knowledge would be incomplete. Through this project we have demonstrated the capabilities of the system to provide the end user with the best product recommendation.

IX. FUTURE WORK

We would like to commercialize the application to be used by users who have limited knowledge of the product. Also, we would like to scale this application from laptops to other electronic products like mobile phones, tablet PCs, Cameras and other complex electronic products where they have too many specifications to be considered while making a purchase.

X. ACKNOWLEDGEMENT

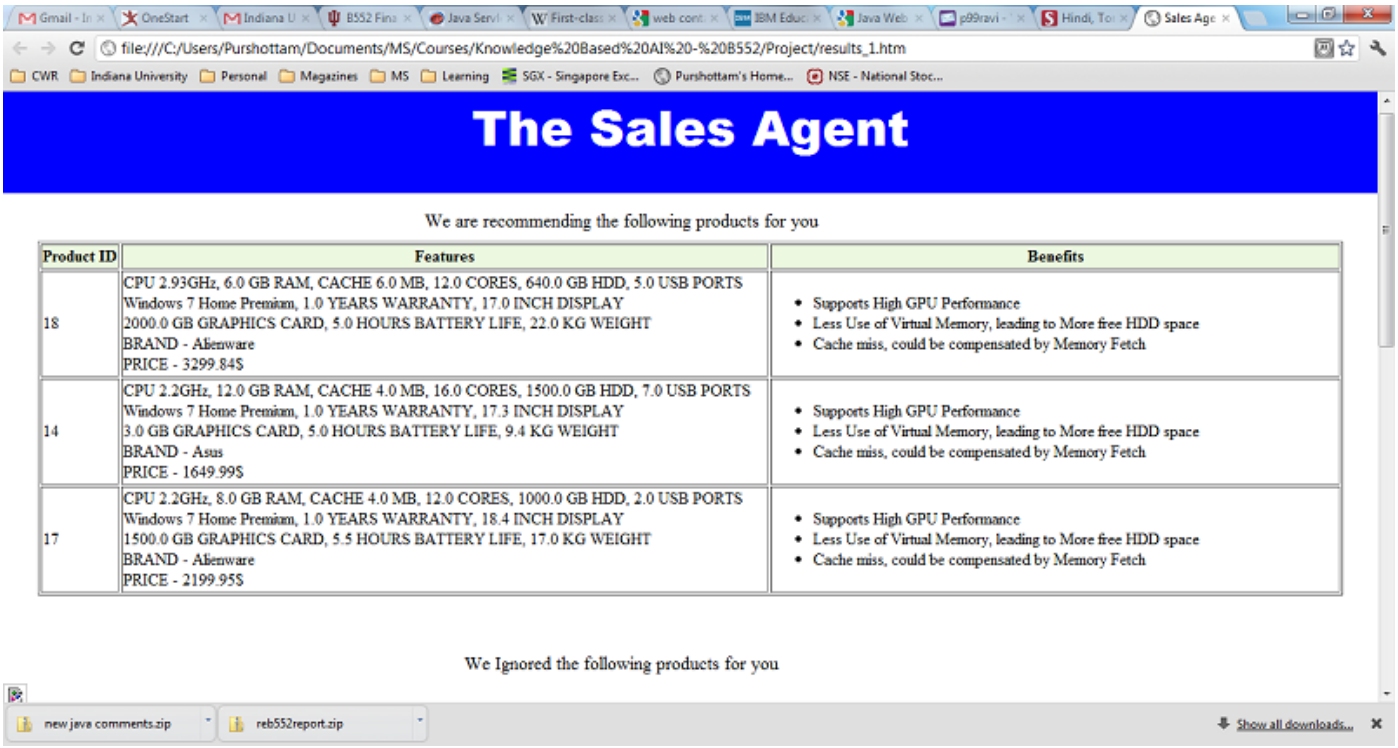
We would like to thank our professor Dr. David Leake and our Associate Instructor Vahid Jalalibarsari for their guidance and assistance in completing this project. This project has given us deep insight and provided the opportunity to explore further into the arena of Artificial Intelligence. I would further like to thank all the fellow students in the B552 class at IU, Bloomington for their valuable discussion and thoughts for this project.

REFERENCES

[1] Leake, D. Case-Based Reasoning: Experiences, Lessons, and Future Directions. AAAI Press, 1996.
[2] Riesbeck, C. and Schank, R. Inside Case-Based Reasoning. Erlbaum, 1989.
[3] Russell, S. and Norvig, P. Artificial Intelligence: A Modern Approach, Third edition, Prentice Hall, 2009.
[4] Schank, R.C., Riesbeck, C., and Kass, A. Inside case-based explanation. Erlbaum, 1994.
[5] Schank, R.C., and Riesbeck, C. Inside Computer Understanding, Erlbaum, 1981.

Appendix A

This is the snapshot of the final results page – List of Recommended Products.



The Sales Agent

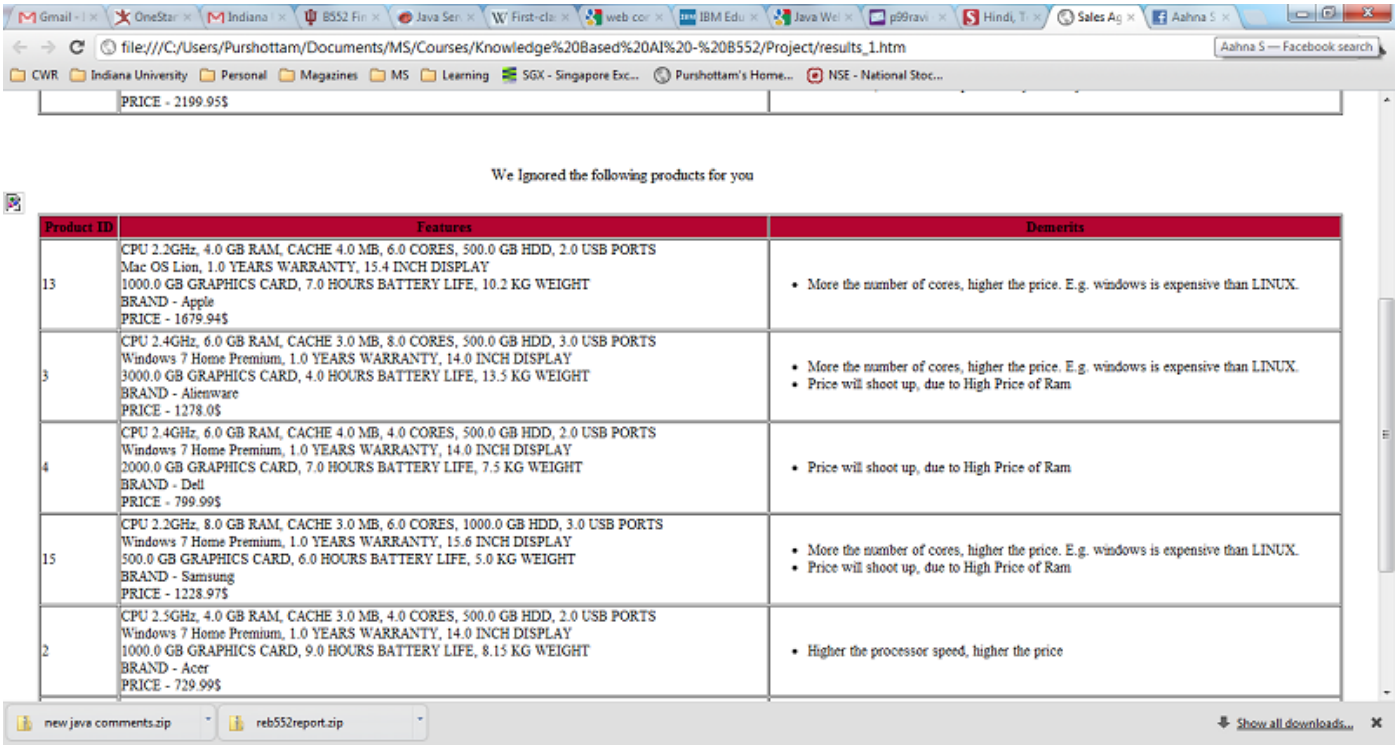
We are recommending the following products for you

Product ID	Features	Benefits
18	CPU 2.93GHz, 6.0 GB RAM, CACHE 6.0 MB, 12.0 CORES, 640.0 GB HDD, 5.0 USB PORTS Windows 7 Home Premium, 1.0 YEARS WARRANTY, 17.0 INCH DISPLAY 2000.0 GB GRAPHICS CARD, 5.0 HOURS BATTERY LIFE, 22.0 KG WEIGHT BRAND - Alienware PRICE - 3299.84$	• Supports High GPU Performance • Less Use of Virtual Memory, leading to More free HDD space • Cache miss, could be compensated by Memory Fetch
14	CPU 2.2GHz, 12.0 GB RAM, CACHE 4.0 MB, 16.0 CORES, 1500.0 GB HDD, 7.0 USB PORTS Windows 7 Home Premium, 1.0 YEARS WARRANTY, 17.3 INCH DISPLAY 3.0 GB GRAPHICS CARD, 5.0 HOURS BATTERY LIFE, 9.4 KG WEIGHT BRAND - Asus PRICE - 1649.99$	• Supports High GPU Performance • Less Use of Virtual Memory, leading to More free HDD space • Cache miss, could be compensated by Memory Fetch
17	CPU 2.2GHz, 8.0 GB RAM, CACHE 4.0 MB, 12.0 CORES, 1000.0 GB HDD, 2.0 USB PORTS Windows 7 Home Premium, 1.0 YEARS WARRANTY, 18.4 INCH DISPLAY 1500.0 GB GRAPHICS CARD, 5.5 HOURS BATTERY LIFE, 17.0 KG WEIGHT BRAND - Alienware PRICE - 2199.95$	• Supports High GPU Performance • Less Use of Virtual Memory, leading to More free HDD space • Cache miss, could be compensated by Memory Fetch

We Ignored the following products for you

Appendix B

This is the snapshot of the final results page – List of Ignored / Omitted Products.



PRICE - 2199.95$

We Ignored the following products for you

Product ID	Features	Demerits
13	CPU 2.2GHz, 4.0 GB RAM, CACHE 4.0 MB, 6.0 CORES, 500.0 GB HDD, 2.0 USB PORTS Mac OS Lion, 1.0 YEARS WARRANTY, 15.4 INCH DISPLAY 1000.0 GB GRAPHICS CARD, 7.0 HOURS BATTERY LIFE, 10.2 KG WEIGHT BRAND - Apple PRICE - 1679.94$	• More the number of cores, higher the price. E.g. windows is expensive than LINUX.
3	CPU 2.4GHz, 6.0 GB RAM, CACHE 3.0 MB, 8.0 CORES, 500.0 GB HDD, 3.0 USB PORTS Windows 7 Home Premium, 1.0 YEARS WARRANTY, 14.0 INCH DISPLAY 3000.0 GB GRAPHICS CARD, 4.0 HOURS BATTERY LIFE, 13.5 KG WEIGHT BRAND - Alienware PRICE - 1278.0$	• More the number of cores, higher the price. E.g. windows is expensive than LINUX. • Price will shoot up, due to High Price of Ram
4	CPU 2.4GHz, 6.0 GB RAM, CACHE 4.0 MB, 4.0 CORES, 500.0 GB HDD, 2.0 USB PORTS Windows 7 Home Premium, 1.0 YEARS WARRANTY, 14.0 INCH DISPLAY 2000.0 GB GRAPHICS CARD, 7.0 HOURS BATTERY LIFE, 7.5 KG WEIGHT BRAND - Dell PRICE - 799.99$	• Price will shoot up, due to High Price of Ram
15	CPU 2.2GHz, 8.0 GB RAM, CACHE 3.0 MB, 6.0 CORES, 1000.0 GB HDD, 3.0 USB PORTS Windows 7 Home Premium, 1.0 YEARS WARRANTY, 15.6 INCH DISPLAY 500.0 GB GRAPHICS CARD, 6.0 HOURS BATTERY LIFE, 5.0 KG WEIGHT BRAND - Samsung PRICE - 1228.97$	• More the number of cores, higher the price. E.g. windows is expensive than LINUX. • Price will shoot up, due to High Price of Ram
2	CPU 2.5GHz, 4.0 GB RAM, CACHE 3.0 MB, 4.0 CORES, 500.0 GB HDD, 2.0 USB PORTS Windows 7 Home Premium, 1.0 YEARS WARRANTY, 14.0 INCH DISPLAY 1000.0 GB GRAPHICS CARD, 9.0 HOURS BATTERY LIFE, 8.15 KG WEIGHT BRAND - Acer PRICE - 729.99$	• Higher the processor speed, higher the price