CHAPTER 4:

KNOWLEDGE PRE-CLASSIFIERS
4.0 Introduction

The summary of the literature review shows that knowledge classification starts with all knowledge contents at the same time, without any ‘staging’. The problems with this aspect of popular knowledge classification approaches are: 1) They do not exploit the semi-structured, limited knowledge content present in subject or headers by combining these text fields with the text bodies or free-flowing document texts. This reduces resource-efficiency and performance of the knowledge classification systems. 2) Using a pre-classifier similar to the concepts of pre-compilers in combined language coding systems, the staged approach of knowledge classification i.e. first identifying a pre-classifier and using it as a basic classification information tag and then analyzing the full text, the efficiency of knowledge classification systems can go up manifold. These observations from the literature review are the main premises or research problems that are solved and are being covered in this chapter.

4.1 A Knowledge Pre-processing Framework

In this section, the first contribution of this thesis is presented as a knowledge pre-processing framework, and is validated by demonstrating its usages and applications using the case-based research methodology.

A knowledge pre-classifier is similar in concept to the idea of a pre-compiler like Pro*C, Pro*Ada etc. that allows different coding language statements to be embedded in Oracle SQL domain. To process these embedded statement, the Oracle run-time environment requires a filter in form of a pre-compiler of Pro*C, Pro*Ada etc. that first filters out and then compiles these non-SQL language constructs and then return the output to the SQL main body.
Following the same principle, in case of knowledge pre-classifier, a pre-processing unit will be used before the entire document or text knowledge body enters the main classification algorithm or system.

However, the main difference between the two contexts is: a) the programming language executable code generation, and b) knowledge pre-processing- is essentially the fact that the output of the knowledge pre-processor is not any executable code etc. but some structured information about the knowledge source that is being input to a knowledge extraction module. The other significant difference which is a basic one is the fact that input for a compiler is a source code file with a specific programming language as using regular expressions and regular grammar, whereas in case of a knowledge pre-processor the input will be free flowing text strings for example as constructs in CFL(Context Free language).

There have already been some applications of compiler-related techniques for discovering classification information from unstructured text, like topic searching using lexical analysis, lexical chains etc. Here, the main purpose is not to extend any of these techniques or even enter into the searching algorithms, pattern search or thesaurus-based pattern matching algorithms which get applied to the entire body of the messages/documents i.e. the whole of the unstructured inputs.

On the contrary, in this thesis, the concept of using a pre-processor is proposed based on similar concept like compilers, along with some explanations and examples of it’s possible use and benefits. Towards this end, some of the approaches first discussed are pattern discovery, subject identification, classification and clustering of unstructured/ semi-structured documents. Then a clean-slate approach is taken with zero assumptions about the concept of knowledge pre-processing, and develop a new generic model for doing the same.
Therefore, the thesis contribution starts from creation of the generic model outline for
knowledge pre-processor which explains the basic framework of the knowledge pre-
processor and its generic components, their roles and inter-relations.

One of the methods for representing documents as networks using partitional and
hierarchical clustering techniques is further explained in this section, to compare it’s strength
and applicability with the proposed knowledge pre-processing model here. This section is
based on the work of He(2001) and Chen(2001). The original research was aimed at
classifying hypertext documents, but the process logic is appealing for applications to any
unstructured text domain. The basics of this process are as follows:

- Any knowledge source/ input is treated unstructured documents
- Co-occurrence (He, 2001) analysis is used to find the similarities and then
  consequently the dissimilarities between the documents. This is done as follows:

  Co-occurrence analysis converts data indices and weights obtained from
  inputs of parameters and various document sources e.g. email/text message
  bodies, into a matrix that shows the similarity between every pair of such

When measured between two documents, say Ei and Ej,

\[
\text{Sim}_{ij} = \alpha \{ A_{ij} / |A|2 \} + \beta S_{ij} / |S|2+ (1 - \alpha - \beta) C_{ij} / |C|2 \]  \[1\]

0 < \alpha, \beta (parameters) < 1, 0 <= \alpha + \beta <= 1,

where A, S, and C are matrices for A_{ij}, S_{ij}, and C_{ij} respectively.

Values for A_{ij} will be 1 if Ei has a direct link/ reference/ hyperlink to Ej, else 0. S is
the asymmetric similarity score Ei and Ej, and is calculated as follows:

\[
p \rightarrow n \]
\[ S_{ij} = \text{sim}(E_i, E_j) = \left[ \sum_{k=1}^{n} d_{ki} d_{kj} \right] / \left[ \sum_{k=1}^{n} d^2_{dij} \right] \]

where \( n \) is total number of terms in \( E_i \), \( m \) is total number of terms in \( E_j \), \( p \) is total number of terms that appear in both \( E_i \) and \( E_j \), \( d_{ij} = (\text{Number of occurrence of term } j \text{ in } E_i) \times \log\left( \frac{N}{df_j} \times w_j \times \text{Termtype factor} \right) \), \( df_j \) is number of documents containing term \( j \); \( w_j \) is number of words in term \( j \); \( \text{Termtype factor} = 1 + \left( \frac{10 - 2 \times \text{type}_j}{10} \right) \), where \( \text{type}_j = \min 1 \text{ if term } j \text{ appears in subject, } 2 \text{ if it appears in body, } 3 \text{ if it appears in ‘note’ etc.} \) and \( C_{ij} \) is number of Es pointing to both \( E_i \) and \( E_j \) (co-citation/ cross-referencing matrix).

- Document bodies which are very similar in terms of their contents i.e. many of the identified key-terms (i.e. Terms excluding the general terms like pro-nouns, prepositions, conjunctions etc.) are same, can be clubbed up together to form a cluster. Dissimilar document bodies can be created as other clusters.

- These clusters can then form a network using hierarchical and partitional clustering method to form a graph with the nodes as representative knowledge maps for a particular group of documents with high-similarity in their body text.

- Partitioning of a graph, say \( G \), can be done in various ways, for example, by using similarity measures as below: (Rich & Knight 2001, Shi & Malik 2000)

\[ \text{Normalized Cut (x)} = \]

\[ \{ \text{cut between } (A, B)/ \text{assoc}(A, V) \} + \{ \text{cut between } (A, B)/ \text{assoc}(B, V) \} \]  

where, \( \text{Cut between } (A, B) = \sum_{i \in A, j \in B} \text{Sim}_{ij} \), \( \text{Sim}_{ij} \) is similarity between nodes \( i \) and \( j \) of the graph. \( \text{Assoc}(A, V) \) and \( \text{assoc}(B, V) \) shows how on average nodes within a group are
connected to each other. A cut on a graph $G = (V, E)$ is defined as removal of a set of edges such that the graph is split into disconnected sub-graphs. (Chen et al 1998, Chen et al 2001). Now, this approach can work fine when the whole document has no element of structure in it at all i.e. any headers / titles / subject lines etc., or these also are combined together along with the body text and are processed together as well, not separately. This property is the main strength as well as weakness of this approach in specific and these kind of clustering-based approaches in general. The strength is that it can handle the whole document as a whole. The weakness is, in doing so, 1) It fails to exploit whatever little structure-related information that is embedded in some part of the document structure itself e.g. label, headings etc., 2) the complex and repetitive nature of the algorithm makes it extremely resource-intensive and in absence of such intensive or dedicated resources, extremely slow. Other approaches like lexical chains suffer from similar constraints. Lexical chains arise from concepts of lexical cohesion that may arise from semantic connections between words (Chali 2005). Deriving the cohesion structure of a text is equivalent to retrieving lexical chains like $LC = \{w_1, w_2, \ldots, w_n\}$. These approaches while working fine with entire text as inputs, as is the case of topic discovery, searching or matching, do not again exploit certain default structured properties of text documents. The concept of LCs however, can be used appropriately within the context of this thesis as well, i.e. users can create the first level of document identifiers or classifiers by applying these LC-discovery concepts to the document label information itself e.g. the heading/subject lines etc. This thesis has actually used the concept similar to that of Roget’s thesaurus as explained by Chali (2005), in the lexical analysis equivalence part of our model.
Generic model outline for knowledge pre-processing and pre-classifier

The generic model of knowledge pre-processor, as explained in the section above, is shown in figure 5 below:

![Diagram of knowledge pre-processor](image)

**Figure 5: Positioning the Knowledge Pre-processor**
The generic model is specified with the knowledge pre-classifier logical design with the required system modules as shown in figure 6 below.

4.2 Explanation of the sub-modules of the knowledge-preprocessing module

- Lexical information extractor: This is designed in line of lexical analyzer in compilers, the main differences being that in case of compilers, the output of a lexical analyzer is a symbol table with tokens, lexemes and patterns. But here the output of a lexical analyzer will be broken-down fragments of the subject sentence into nouns/verbs/adjectives/adverbs etc. (the identification of a noun/verb and its subgroups e.g. names/objects/functions etc. can be done by using pattern matching and thesaurus). If we represent this analogy as in figure 6, we get the symbol table equivalent in knowledge pre-processor as shown in table 1 inside figure 6:
Syntactic information extractor: The syntax extractor can draw its equivalence to the syntax directed definitions including the annotated parse trees, dependency graphs, evaluation order-based graphs and syntax trees. A syntax tree can be thought as a condensed form of parse tree useful for representing language constructs. For example, a production rule-type knowledge presentation scheme can appear as a syntax tree in the following form:

\[
\text{If-} \quad \text{then-} \quad \text{else}
\]

\[
B \quad S1 \quad S2
\]

For a production rule: If B then S1 else S2.
Semantic information extractor: Can be designed with an equivalence of semantic analyzer. The output can take form of a semantic network.

In the following section, we use a simple example drawn from a practical application situation and take this example through the initial steps in the knowledge pre-processor, basically onto up to the lexical analyzer equivalent part. This example can be further worked upon for generating the syntax trees as explained briefly above. And then it can be taken further to form it’s semantic net equivalent.

An example:

Suppose there are two customer e-mail messages about trouble-shooting, to be input to a CRM knowledge base. The email messages have their subject lines in a fairly structured fashion, as they have used the pre-defined form fields of customer feedback forms on the company websites. These subject fields are considered as two inputs strings in this example. They are as follows:

| InputString1: {Microwave model no. 2021 purchased in year 2002 not functioning: the table is not rotating} |
| InputString2: {Microwave model no. 4576 purchased in year 2005 not functioning: heating is not proper} |

First level of lexical analysis on these two strings may generate output as follows:

| InputString1: [Microwave model no. 2021 purchased in year 2002 not functioning]: (considered as connector) [the table is not rotating] |
InputString2:[Microwave model no. 4576 purchased in year 2005 not functioning]: [heating is not proper]

2nd level: nouns (match from dictionary of nouns: can be made restricted to contexts: e.g. names(e.g. in case of customers complaining about service etc. by names), objects (as microwave in this example), place-names, function-names(e.g. ‘heating’ in the 2nd input string and so on)

InputString1: [[Microwave] [model no. 2021] purchased in [year 2002] not functioning] : [the [table] is not rotating]

InputString2: [[Microwave] [model no. 4576] purchased in [ year 2005] not functioning]: [heating is not proper]

3rd level: verbs

InputString1: [[Microwave] [model no. 2021] [purchased] in [year 2002][ not functioning]] : [the [table] is [not rotating]]

InputString2: [[Microwave] [model no. 4576] [purchased] in [ year 2005] not functioning]: [heating] is [not proper]

4th level: qualifiers/ adjectives

InputString1: [[Microwave] [model no. 2021] [purchased] in [year 2002][ not functioning]] : [the [table] is [not rotating]]

InputString2: [[Microwave] [model no. 4576] [purchased] in [ year 2005] not functioning]: [heating] is [not proper]]

Now, suppose we construct a table to store these strings as analyzed by the lexical extractor we get:
<table>
<thead>
<tr>
<th>Type of string component</th>
<th>Noun</th>
<th>Verb</th>
<th>Adjectives</th>
<th>Adverbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputString1: substring1</td>
<td>[Microwave]</td>
<td>[purchased]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[model no. 2021]</td>
<td>[not functioning]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[year 2002]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InputString1: substring2</td>
<td>[table]</td>
<td>[not rotating]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InputString2: substring1</td>
<td>[Microwave]</td>
<td>[purchased]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[model no. 4576]</td>
<td>[not functioning]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[year 2005]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InputString2: substring2</td>
<td>[heating]</td>
<td></td>
<td>[not proper]</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: A minimal view of the lexical extractor output with two example input strings

This table can be further fine-tuned, for example, by using a look-up table with index values for all these word-types and their sequential combinations e.g. 1 for nouns, and then 11 for names, 12 for objects, 13 for verb-type nouns e.g. function-names (like ‘heating’), 2 for verbs(21 for auxiliary verbs, 22 for continuous tense …), 3 for adjectives, 4 for binary(yes/
no-not) response and so on. So, a phrase like ‘Heating is not proper’ can be expressed using this preliminary look-up table would be

\[
\begin{array}{cccc}
<13> & <21> & <4> & <3>
\end{array}
\]

[Heating] [is] [not] [proper]

This whole string can be stored as an identifier with the numbers as indices for specific values as 13-21-4-3, just to remember the structure of the phrase. This information can be further added as the syntactic information for the phrases which would help in easy reconstruction of the phrases and subsequently easy and highly understandable retrieval.

Also, the connectors may give valuable information, e.g. in this example case the symbol ‘:’ depicts a further explanation of the problem, whereas in other cases the same symbol might mean different things e.g. cause-and-effect link between the two constructs. So, the connector along with its semantic role as a connector (e.g. a further explanatory/ a cause-and-effect link) will also have to be stored as part of the semantic extractor’s job.

The rest of the example can be worked upon using further concepts on syntax and semantic analysis, as has already been mentioned before. Also, we can combine this model with the LC or Co-occurrence analysis models as explained in earlier sections and can make the process more efficient.

This presents a fresh approach for knowledge extraction from unstructured sources using the concept of a pre-processor and the tried and tested concepts of traditional compiler construction in theoretical as well as applied computer sciences domain. The primary advantage of having a knowledge pre-processor, as has been explained in the first section of this thesis, is the fact that a pre-processor can perform a level 0 analyzing and discover or present a basic identifier or classifier for an unstructured knowledge source by exploiting
some amount of structured string-type information that are usually present in the source headers or document labels or message subjects/headings. This way it can reduce the workload of a knowledge extraction module which can then take the entire body-text of the document/message/knowledge source and apply the well-researched approaches of unstructured text handling on them. This way the entire process of knowledge extraction becomes faster and more resource-efficient. Further research possibilities include detailed design and implementation of the sub-modules under the knowledge pre-processor and exploiting the opportunities there again to use the tired and tested concepts of compilers, theory of computer science, theory of languages like regular grammar and CFL etc. With reference to the model presented in this thesis, there are research issues in terms of scalability of the model e.g. the volume of unstructured data as well as heterogeneous source support-systems that can be handled by the model. Also there are issues related to the implementation, performance, resource utilization and tuning of any system based on this model which includes questions like which algorithms to choose for unstructured information handling, topic detection, preliminary information extraction, clustering etc., how to optimize the resource utilization for these algorithms, how to improve performance of an actual knowledge preprocessor and so on. Therefore, the model presented in this thesis can be extended in multiple dimensions including theoretical aspects like algorithms design and analysis to implementation aspects including scalability and performance issues.

4.3 Dependency Graph-based Knowledge Pre-processor and Pre-Classifier Framework

In the previous section, the knowledge pre-classifier framework is built on the premises equivalent to the working principles of pre-compilers. In this section, another approach
based on graph theoretical constructs—primarily, dependency graphs, incremental computing and transitive closures, is developed and validated using case applications.

### 4.3.1 Basic concepts of Graph theory and dependency graphs

Given a set of objects $S$ and a transitive relation $R = S \times S$, and $a, b$ belong to $R$, a dependency that $a$ needs to be evaluated before $b$ is a dependency graph of $a \rightarrow b$. In a dependency graph, cycles of dependencies or circular dependencies can lead to a situation in which no valid evaluation order can be found as none of the objects in the cycle may be evaluated first. If a dependency graph does not have circular dependencies, and resembles a directed acyclic graph, in that case an evaluation order can be found by topological sorting. Most topological sorting algorithms are also capable of detecting cycles in their inputs.

For a dependency graph without circular dependencies, the dependency diagram can be established as a Hasse diagram of the graph. Dependency diagrams are appropriate tools for outlining the complex, interrelationships of various functional elements. Typically in a dependency diagram, arrows point from each module to other modules which they are dependent upon.

Use of dependency graphs is one of the fundamental principles of various software design especially systems software, compilers and parsers, code generators or interpreters or utilities design where the application dependencies are critical to determine correct evaluation and running orders and efficient resource utilization without deadlocks. These have been used extensively in automated software installers that walk the graph looking for software packages that are required but not yet installed. The dependency is given by the coupling of the packages. Software build scripts such as Unix Make, Apache Ant use dependency graphs to find out what files have changed so only the correct files need to be recompiled. In
application software, Web Forms standards such as XForms, Spreadsheet calculators and Dead code elimination modules work on the principles of dependency graphs.

4.3.2 Incremental Computing

It is a software feature also known as incremental computation which attempts to save time by re-computing only the outputs that depend on or are affected by incrementally changed data. An incremental computing system typically has a predefined smallest unit of change that will be individually tracked. If a change is made that is smaller in scope than this smallest unit, the containing unit will be deemed to have changed.

4.3.3 Transitive Closures

In mathematics, the transitive closure of a binary relation $R$ on a set $X$ is the smallest transitive relation on $X$ that contains $R$. For any relation $R$, the transitive closure of $R$ always exists. In graph theory, any binary relation $R$ on a set $X$ may be thought of as a directed graph $(V, A)$, where $V = X$ is the vertex set and $A = R$ is the set of arcs of the graph. The transitive closure of a directed acyclic graph or DAG is the reachability relation of the DAG and a strict partial order.

In computer science the concept of transitive closure can be thought of as constructing a data structure that makes it possible to answer reach-ability questions. That is, can one get from node $a$ to node $d$ in one or more hops? A binary relation tells only that node $a$ is connected to node $b$, and that node $b$ is connected to node $c$, etc. After the transitive closure is constructed, as depicted in the following figure, in an $O(1)$ operation one may determine that node $d$ is reachable from node $a$. The data structure is typically stored as a matrix, so if matrix $1][4] = 1$, then it is the case that node 1 can reach node 4 through one or more hops. The transitive reduction of a graph is sometimes referred to as its minimal representation.
The transitive reduction of a finite acyclic graph is unique. For a graph with nontrivial strongly connected components, each such component will become a cycle in any transitive reduction of that graph.

In order theory, a branch of mathematics, a Hasse diagram is a simple picture of a finite partially ordered set, forming a drawing of the transitive reduction of the partial order. Concretely, for a partially ordered set \((S, \leq)\) one represents each element of \(S\) as a vertex on the page and draws a line segment or curve that goes upward from \(x\) to \(y\) if \(x \leq y\), and there is no \(z\) such that \(x < z < y\) (here, \(<\) is obtained from \(\leq\) by removing elements \((x, x)\) for all \(x\)).

Knowledge extraction from semi-structured or unstructured documents and texts are handled by various approaches and viewpoints, including theoretical or mathematical constructions and matrix algebra applications to various ontologies and implementation techniques. However, unstructured knowledge sources also have limited amount of knowledge elements which can be used for preliminary classification of these sources, making it easy for further analysis. Such knowledge elements are embedded in the headers or subject lines of these knowledge sources.

In this thesis, a new approach has been proposed where these headers or subject lines can be analyzed first, to extract preliminary classification elements from them, using the dependency graph-based concepts, which is also explained in the thesis.

Extracting reusable knowledge elements from unstructured sources has been a cross-functional research area for many years now. This finds research and applications both in purely theoretical concept-building efforts like mathematical and statistical analysis (Beeferman, Berger, Lafferty (1999)) of documents for similarity and dissimilarity, using matrices and various co-efficient and formulae, as well as purely applied domains which are
promoted by organizations like IBM (ref. UIMA – Unstructured Information Management Architecture proposed by IBM). These practical approaches focus mainly on various ontologies and their implementation. In between these, there have been significant progress in algorithms, application of various NLP (Natural language Processing) concepts and other artificial intelligence-based algorithms like genetic algorithms, neural networks and pattern recognition/ pattern matching logic and systems.

As can be identified from the contemporary research directions, there are few specific research issues related to unstructured/ free-flowing text, which make it an interesting domain for researchers with purely theoretical, purely practical or mixed orientations. The issues range from highly theoretical, mathematical, logical and analytical dimensions like discovering cohesions and relations between various sections of body texts (e.g. paragraphs), discovering topics, searching for topics. Further issues are related to the practical or implementations-specific side of the problem e.g. storing the discovered/ searched information in a knowledge representation format which is more accessible, understandable, implement-able and easily retrievable to achieve the ultimate goal of re-usable knowledge repositories. These issues translate down to specific research questions like: text segmentation, topic tracking, topic detection, link detection, classification and clustering.

It is seen that actually many of the knowledge sources which are generally viewed as ‘unstructured’ or ‘free-flowing texts’ can be found to have some degree of explicit structuredness for example embedded in their labels. Unfortunately, these already embedded ‘semi-structured’ information which can help any extraction module to do some preliminary classification, do not get adequately exploited if the whole document along with the semi-structured part also is input at the beginning itself to the extraction modules.
For example, there can be limited amount of information embedded or available in the document headers, message headings, subject lines of letters or emails and so on. These, if adequately processed before entering into the actual full-document analysis-based extraction phase, some classification information can already be made available to the extraction modules.

This limited amount of structured information that are usually present in unstructured textual knowledge sources in their headers or subject lines can be used as shown below, for a preliminary classification of these sources.

A dependency graph, as mentioned in previous section, shows the order of dependencies between objects. Objects can be events, data objects, service objects, parent-child dependency-based objects or an interconnection between them, e.g. event -> data dependency or a particular value of data>threshold value => triggers an event: dependency, and so on.

For example, if there is a simple calculator application, that would support assignment of constant values to variables and assigning the sum of exactly two variables to a third variable. Given several equations like “A = B+C; B = 15+D; C=24; D=20;”, then S, the complete data-set = A,B,C,D and R, the dependency graph sets = [(A,B),(A,C),(B,D)], as A depends on B and C (one can add two variables if and only if the values of both variables are known). Thus, B and C must be calculated before A can be calculated. D’s value is known immediately, because it is a number literal. Similarly, the equation system [A=B; B=D+C; C=D+A; D=20] contains a circular dependency formed by A, B and C, as B must be evaluated before A, C must be evaluated before B and A must be evaluated before C. This is primarily a data-operations dependency scenario.
Dependency graphs have been widely used in compiler construction too, thus having a proven track record in terms of efficiency and completeness. The following chain as shown below explains the basic role of a dependency graph in the translation of languages guided by context-free grammars.

With syntax-directed definitions and translation schemes, the input token stream can be first parsed for constructing the parse tree. Then this parse-tree can be traversed as required for evaluating the semantic rules at the nodes. This process can give various types of outputs e.g. code, information in a symbol table, error messages etc. Finally the translation of the token stream is the result of the evaluation of the semantic rules which are reflected as nodes in the parse tree.

In the context of a compiler, semantic rules set up dependencies between the attributes represented by a graph, which can be in the form of an annotated parse tree, which shows the values of the attributes at each node. (“The process of computing the attribute values at the nodes is called annotating or decorating the parse tree” – Aho et al, 2001). In a syntax-directed definition, which is the most common case for programming language constructs and hence compilers, each grammar production $A \rightarrow \alpha$ is associated with a set of semantic rules of the form $b := f(c_1, c_2, \ldots, c_k)$ where $f$ is a function and $b$ is either a synthesized attribute of $A(c_1, c_2, \ldots, c_k$ belonging to grammar symbols of the production) or $b$ is an inherited attribute of one of the grammar symbols on the RHS of the production.

If an attribute $b$ at a node in a parse tree depends on an attribute $c$ (which is generally the typical case for a natural language construct), then the semantic rule for $b$ at that node must be evaluated after the semantic rule which defined $c$, to preserve the dependency relationships between $b$ and $c$. These interdependencies among the inherited and synthesized attributes at
the nodes in a parse tree are depicted by a directed graph. This is called a dependency graph.
(ref. figure 1 example)

Any topological sort of a dependency graph gives a valid order for evaluating semantic rules in a parse tree. A topological sort of a direct acyclic graph is defined as “any ordering m1, m2, ..., mk of the nodes of the graph such that edges go from nodes earlier in the ordering to the later nodes” (Aho et al 2001), i.e. if mi ->mj is an edge from mi to mj then mi appears before mj in the ordering.

This means that, in the topological sort on a dependency graph, the dependent attributes c1, c2, ..., ck in a semantic rule b:= f (c1, c2, ..., ck ) are available at a node before f is evaluated. This evaluation rule is applied equivalently in this thesis in the following section in the context of an example.

The application of dependency graph in this thesis, however, is not exactly based on its role in case of programming language-based constructs. The differences are:

- The primary difference here is that in case of programming language constructs, there are various operators e.g. arithmetic, logical, Boolean operators etc. and also there are various keywords identifying data types(e.g. int, char, string, Boolean) or processing logic (e.g. {if...then}, {while{}}, {do while{}} constructs etc. But in the context of natural language-based sentences these have no particular significance. But natural language e.g. English also essentially follows certain grammatical rules and syntax which govern the basic construction of sentences. For example, if we take the e-mail message headings or customer complaint headings, most often they are not grammatical correct and complete sentences. But they follow certain rules, e.g. in ‘sub’ (subject) header it talks about the basic topic of the e-mail or message and in not
self-explanatory, then from whom it is coming (source identified) and to whom it should reach (‘attn:’) etc.

- From the first point, the second difference gets visible that in case of programming languages, the full source code is in for analysis, whereas in this thesis, the objective is to extract as much classification and categorization information as possible from the one-sentence constructs present in the labels or headers or subject-part of unstructured communication/ document elements like e-mail messages, web documents, attachments etc.

The application of dependency graphs is explained in this thesis through an example of e-mails as the knowledge source inputs. However, the logic can be extended and applied to any generic unstructured document context as long as it has a label/ heading/ subject-line/ document name.

The first difference as mentioned above gets addressed in this thesis by the inclusion of a Thesaurus. In this case a mention of Roget’s thesaurus as has been proposed by Chali (2005) is worthwhile. As opposed to the case of a programming language compiler which deals with keywords and symbols and strings for operators, operations, data types and variable names, a preliminary knowledge extractor using the dependency graphs would need a thesaurus to look up for the values at each node of the parse tree in form of words and thereafter evaluate the semantic rules presented by the parse tree. The following example illustrates this:

An example complaint message (posted on a company portal’s ‘Contact us’ form) header:
“reporting: employee XYZ not able to answer my query”- from customer no. 897, complaint no. 67, timestamp dd/mm/yyyy-mm-hh.
This sentence can be broken down in form of an equivalent of an LR parse tree, as shown in figure 8 below:

Reporting: employee XYZ not able to answer my query

![Parse Tree Diagram]

**Figure 8: An Example of a Basic Parse Tree**

The parse tree shows the attributes specific to the input sentence along with the type of the attribute given in italics below the attribute labels. For interpreting these attributes, a thesaurus is needed.

In the parse tree as shown above, two basic semantic rules can be identified as follows (in the form equivalent to $b: = f(c_1, c_2, \ldots, c_k)$ as explained in the previous section)

1. $b_1 := \text{about} \left( \text{query}(\text{query no.}(\text{value}), \text{timestamp}(\text{value})) \right)$

2. $b_2 := \text{not} \left( \text{answering} \right)$

If these rules are shown in the parse tree, the dependency graph can be viewed as in figure 9:
From the topological sort of a dependency graph, an evaluation order for semantic rules can be obtained. For example: the evaluation order as found from a simple topological sort on the graph above is: b1 -> b2. Evaluation of the semantic rules e.g. b1 and b2 in this order yields the translation of the input string reporting: employee XYZ not able to answer my query—from customer no. 897, complaint no. 67, timestamp dd/mm/yyyy-mm-hh.

This example query can thus be initially classified using a dependency graph, as a ‘query not answered’ class of problems and this initial classification information can be used for categorizing this snippet of customer experiential knowledge into the services knowledge-base of the organization.

This is a clean-slate approach as well as a new concept for knowledge extraction from unstructured sources using the concept of a preliminary classification using the tried and tested concepts of traditional compiler construction in theoretical as well as applied computer sciences domain. The primary advantage of the approach is the fact that it can present a basic identifier or classifier for an unstructured knowledge source by exploiting some amount of structured string-type information that are usually present in the source headers or document.
labels or message subjects/headings. This can reduce the workload of a knowledge extraction module which can then take the entire body-text of the document/message/knowledge source and apply the well-researched approaches of unstructured text handling on them. This way the entire process of knowledge extraction becomes faster and more resource-efficient.

Further research possibilities include applications and extensions of unstructured text handling approaches using other theories in the concepts of compilers, theory of computer science, theory of languages like regular grammar and CFL etc.

In case of semi-structured knowledge, however, most of the concepts of handling unstructured knowledge is extended, as they are highly applicable for semi-structured knowledge too.

But this induces inefficiency and suboptimal computing resource utilization by the algorithms, as the “structured-ness” available to some degree in these semi-structured knowledge sources is not adequately exploited. For example email messages, blogs, essays and articles are considered free-flowing texts and a number of algorithms have been suggested for discovering the key-word that can detect the ‘topic’ and identify certain key-words that are necessary for classification of these texts. But each one of these knowledge sources already have some structured information in most cases- in their headers or subject-lines etc. When we include them in the free-flowing text body, the first opportunity to exploit the hint on it’s likely contents is lost, as the semi-structured information in the short headers or subject-lines are not used as a possible source of topic identification and classification first. These are the semi-structured short texts that, when used efficiently for topic identification or classification can significantly reduce the computing resource requirements and can make knowledge extraction from these unstructured resources more efficient and faster.
In this thesis, a process model has been suggested and explained, using the concepts of
dependency graphs and incremental computing, that can work on the subject-lines or
message-headers first and can act as a ‘pre-classification filter’ for every unstructured
knowledge source. Using incremental computing concepts like transitive reduction and
incremental data structures, the ‘pre-classifier information label’ can be created and that label
can be used during the detailed, exact classification stage. This label can significantly
increase the speed of topic detection and classification algorithms by providing them the
preliminary classification information.

However, common thesaurus-based approaches suffer with one significant efficiency issue in
terms of their absolute neutrality towards handling application-context-specific terms or
words. For example, in a banking CRM communication thread, the text may contain specific
terms like ‘account’ (which is a financial product-related data/term that can be better
understood with a connection to the Bank’s product master records). But a common
thesaurus-based analyzer would take it by its ‘dictionary’ meaning per se, not as it appears in
this specific application context.

Methods for representing documents as networks by using partitional and hierarchical
clustering techniques had been developed by several researchers e.g. He(2001) and
Chen(2001). The original research was aimed at classifying hypertext documents, but the
process logic is appealing for applications to any unstructured text domain. However, this
approach again can work fine when the whole document has no element of structure in it at all
i.e. any headers / titles / subject lines etc., or these also are combined together along with the
body text and are processed together as well, not separately. This property is the main
strength as well as weakness of this approach in specific and these kind of clustering-based
approaches in general. The strength is that it can handle the whole document as a whole. The weakness is, in doing so, 1) It fails to exploit whatever little structure-related information that is embedded in some part of the document structure itself e.g. label, headings etc., 2) the complex and repetitive nature of the algorithm makes it extremely resource-intensive and in absence of such intensive or dedicated resources, extremely slow.

Other approaches like lexical chains suffer from similar constraints. Lexical chains arise from concepts of lexical cohesion that may arise from semantic connections between words (Chali 2005). Deriving the cohesion structure of a text is equivalent to retrieving lexical chains like $LC = \{w_1, w_2, \ldots, w_n\}$.

Summarily speaking, most of these approaches while working fine with 1) entire text as inputs and where 2) in case of topic discovery, searching or matching the input text is fairly generic in nature i.e. no context-specificity of domain-knowledge-value in particular, is present in the text that can be handled separately.

That is how most of the available text analysis tools 1) do not again exploit certain default structured properties of text documents, and 2) being thesaurus-based, the contextual/application terminologies or details that should be treated separately, are also mixed up in the text-under-analysis.

These two research gaps as identified in sections I and II are the primary premises of this model. With the research gaps identified, it is seen that actually many of the knowledge sources which are generally viewed as ‘unstructured’ or ‘free-flowing texts’ can be found to have some degree of explicit structured-ness for example embedded in their labels. Unfortunately, these already embedded ‘semi-structured’ information which can help any extraction module to do some preliminary classification, do not get adequately exploited if the
whole document along with the semi-structured part also is input at the beginning itself to the extraction modules. Almost always there is limited amount of information embedded or available in the document headers, message headings, subject lines of letters or emails and so on. These, if adequately processed before entering into the actual full-document analysis-based extraction phase, some classification information can already be made available to the extraction modules. This thesis proposes and explained through example trial runs, one such process model that can create a pre-classifier information label for unstructured sources, based on their subject-lines or headers. This process model is based on concepts of incremental computing and dependency graphs. This model can be further extended to a model where the application-context or domain-specific terms are better understood, treated and analyzed with respect to not just a common language thesaurus but also domain-specific information available, i.e. the product master data, customer master data and so on, for specific, contextually meaningful terms and their usages.

Incremental computing or incremental computation is a computational method that attempts to save time by re-computing only those outputs which "depend on" a changed data-whenever a piece of data changes. For example, a spreadsheet application uses incremental computation in its recalculation feature, to only update those cells containing formulas which are directly or indirectly linked or dependent on the changed cells. An incremental computing system typically has a predefined smallest unit of change that will be individually tracked. If a change is made that is smaller in scope than this smallest unit, the containing unit will be deemed to have changed. For example, if just one numerical digit of a seven-digit number in a cell in a spreadsheet is altered, the whole cell will be treated as changed as for a spreadsheet the smallest unit is a cell. Incremental compilers address the problem of having to recompile
an entire compilation unit of a program if any of the source files the unit depends on have changed. The smallest units of change for incremental compilers are typically the functions defined in the source program in the source language.

A typical implementation technique for incremental computing is for the software to build a dependency graph of all the data elements that may need to be recalculated, and their dependencies. The transitive closure of dependency relation of the graph is given by the set of elements that need to be updated when a single element changes. That is, if there is a path from the changed element to another element, the latter needs to be updated. Basic definition of transitive closures says that the transitive closure of a binary relation R on a set A is the smallest transitive relation on A that contains R.

For example, if A is a set of employees and xAy means "employee x works for employee y", then the transitive closure of A on X is the relation "it is possible that more employees can work for more of other employees."

The dependency graph may need to be updated as dependencies change, or as elements are added to, or removed from, the system. It is used internally by the implementation, and does not typically need to be displayed to the user.

Dependency graphs are widely used concepts in compiler construction, thus having a proven track record in terms of efficiency and completeness. The chain as shown below explains the basic role of a dependency graph in context of compliers, in the translation of languages guided by context-free grammars.

Input string   | Parse Tree  | Dependency graph | Semantic rules |

Figure 10: Role of dependency graph in context-free constructs analysis (based on Aho, Sethi, Ullman, 2001)
As shown in the diagram, the input token stream with syntax-directed definitions and translation schemes can be first parsed for constructing the parse tree. Then this parse-tree can be traversed as required for evaluating the semantic rules at the nodes. This process can give various types of outputs e.g. code, information in a symbol table, error messages etc. Finally the translation of the token stream is the result of the evaluation of the semantic rules which are reflected as nodes in the parse tree.

In the context of a compiler, semantic rules set up dependencies between the attributes represented by a graph, which can be in the form of an annotated parse tree, which shows the values of the attributes at each node. (“The process of computing the attribute values at the nodes is called annotating or decorating the parse tree” – Aho et.al, 2001). In a syntax-directed definition, which is the most common case for programming language constructs and hence compilers, each grammar production $A \rightarrow \alpha$ is associated with a set of semantic rules of the form $b := f(c_1, c_2, \ldots, c_k)$ where $f$ is a function and $b$ is either a synthesized attribute of $A$($c_1, c_2, \ldots, c_k$ belonging to grammar symbols of the production) or $b$ is an inherited attribute of one of the grammar symbols on the RHS of the production.

If an attribute $b$ at a node in a parse tree depends on an attribute $c$ (which is generally the typical case for a natural language construct), then the semantic rule for $b$ at that node must be evaluated after the semantic rule which defined $c$, to preserve the dependency relationships between $b$ and $c$. These interdependencies among the inherited and synthesized attributes at the nodes in a parse tree are depicted by a directed graph. This directed graph itself is a dependency graph.

The generic construction process of a dependency graph is as follows:
For each node $n$ in the parse tree do

For each attribute $a$ of the grammar symbol at $n$ do

    Construct a node in the dependency
graph for $a$;

For each node $n$ in the parse tree do

    For each semantic rule $b := f(c_1, c_2, \ldots, c_k)$
    Associated with the production used at $n$ do

        For $i := 1$ to $k$ do

            Construct an edge from the node for $c_i$ to the node for $b$;

Any topological sort of a dependency graph gives a valid order for evaluating semantic rules
in a parse tree. A topological sort of a direct acyclic graph is defined as “any ordering
$m_1, m_2, \ldots, m_k$ of the nodes of the graph such that edges go from nodes earlier in
the ordering to the later nodes” (Aho et al 2001), i.e. if $m_i \rightarrow m_j$ is an edge from
$m_i$ to $m_j$ then $m_i$ appears before $m_j$ in the ordering. In the topological sort on a
dependency graph, the dependent attributes $c_1, c_2, \ldots, c_k$ in a semantic rule $b := f(c_1, c_2, \ldots, c_k)$ are available at a node before $f$ is evaluated.

Partial evaluation can be seen as a method for automating the simplest possible case of
incremental computing, in which an attempt is made to divide program data into two
categories: that which can vary based on the program's input, and that which cannot (and the
smallest unit of change is simply "all the data that can vary"). Partial evaluation can be
combined with other incremental computing techniques.

Limited amount of semi-structured knowledge that is embedded in the subject-lines of
unstructured text-bodies like emails, can be utilized for a preliminary classification, using the
incremental computing concepts. The advantages of using the limited amount of information can be many. It can help the knowledge extraction modules more efficient by making a preliminary classification information available to them. These modules are often extremely resource-hungry and slow (due to less availability of such computational resources). Consequently, the knowledge extraction modules tend to become slow because of their unavoidable and extreme logical and processing complexities. A pre-processed input can make the logic simpler to some extent.

It also helps the knowledge extraction modules to exploit some amount of structured information that remained embedded in part of unstructured documents like headings etc.

There have already been some applications of compiler-related techniques for discovering classification information from unstructured text, like topic searching using lexical analysis, lexical chains etc. The main purpose of this thesis is not to extend any of these techniques in themselves or enter into the searching algorithms, pattern search or thesaurus-based pattern matching algorithms which get applied to the entire body of the messages/documents i.e. the whole of the unstructured inputs. On the contrary, a clean-slate approach with zero assumptions about the concept of preliminary knowledge extraction has been taken in this thesis. Precisely speaking, the thesaurus-based approaches have been used here too, but they alone are insufficient in creating application-specific/domain-context-specific ‘information labels’ or ‘knowledge tags’ like pre-classifiers. Therefore, in addition to thesaurus-based approaches, domain/application-context-specific term-bases are also connected to the preliminary classifier, to help the knowledge extraction module to identify the application-context of a particular text-body. For example, ‘complaints’ in case of a CRM application in a bank has a domain and application specificity that is beyond just the dictionary-meaning.
However, the main difference between the two contexts i.e. a compiler’s main function vs. preliminary knowledge extraction as proposed in this thesis, is essentially the fact that the output of the preliminary knowledge extraction process is not any executable code (as is the case with compilers) etc., but an ‘information label’, a ‘knowledge tag’ or a ‘message stamp’ that are: some structured information about the knowledge source (text) that is being input to a knowledge extraction module.

The other significant difference which is a basic one is the fact that input for a compiler is a source code file with a specific programming language as using regular expressions and regular grammar, whereas in case of a knowledge pre-processor the input will be free flowing text strings for example as constructs in CFL(Context Free language). This point has been further elaborated specifically for the application of dependency graphs.

The application of dependency graph in this thesis is not exactly based on its role in case of programming language-based constructs. The differences are:

- In case of programming language constructs, there are various operators e.g. arithmetic, logical, Boolean operators etc. and also there are various keywords identifying data types(e.g. int, char, string, Boolean) or processing logic (e.g. {if…then}, {while{)), {do while{}} constructs etc. But in the context of natural language-based sentences these have no particular significance. But natural language e.g. English also essentially follows certain grammatical rules and syntax which govern the basic construction of sentences.

For example, if we take the e-mail message headings or customer complaint headings, most often they are not grammatical correct and complete sentences. But they follow certain rules, e.g. in ‘sub’ (subject) header it talks about the basic topic of the e-mail or...
message and in not self-explanatory, then from whom it is coming (source identified) and
to whom it should reach (‘attn:’) etc.

- From the first point, the second difference gets visible that in case of programming
  languages, the full source code is in for analysis, whereas in this thesis, the objective is to
  extract as much classification and categorization information as possible from
  the one-phrase constructs present in the headers or subject-lines of unstructured
  communication/ document elements like e-mail messages, web documents, attachments etc.

The application of dependency graphs are explained in this thesis through an example of e-
mails as the knowledge source inputs. However, the logic can be extended and applied to any
generic unstructured document context as long as it has a label/ heading/ subject-line/
document name.

The first difference as mentioned above gets addressed in this thesis by the inclusion of a
Thesaurus. In this case a mention of Roget’s thesaurus as has been proposed by Chali (2005)
is worthwhile. As opposed to the case of a programming language compiler which deals with
keywords and symbols and strings for operators, operations, data types and variable names, a
preliminary knowledge extractor using the dependency graphs would need a thesaurus to look
up for the values at each node of the parse tree in form of words and thereafter evaluate the
semantic rules presented by the parse tree.

A generalized model for this approach is presented in figure 11 below.
Figure 11: Generalized model of a dependency-graph based approach for extracting preliminary classification knowledge elements from unstructured source-headers

This generalized model as shown in figure 11 has been developed based on the basic approaches that are described in the thesis using an example of an application. The possible output of the example used in the thesis is given in figure 11. Now the process model dry-run will be explained in the following sections, to convert a customer complaint input into this example output ‘information label’ or an operational ‘knowledge tag’. The specific application instances of this generic model can be in various organizational contexts, as the
one that is now illustrated in next section of this thesis. For example, extracting re-usable knowledge elements from heterogeneous communication domains including telephonic conversations, chat messages, bulletin boards, groupware messages etc. can be facilitated by this model where the subject header information of these heterogeneous sources can be used for preliminary classifications. Similarly, based on application contexts, the classification schemes can be focusing on functional domains e.g. CRM, new product development, market research – qualitative analysis etc.

An example run with an unstructured email message with a header is given here, to explain the application of the process model as described in figure 1, in a real-life context. (inputs from a standard customer feedback/ grievance form on any standard e-banking portal)

<Email form>

<subject or header>

“Monthly e-statement for s/b a/c no. 567720192 not received in June 2009”

</subject or /header>

<Relationship no. or account no.> s/b a/c no. 567720192: Rony Wilson

</Relationship no. or account no.>

<Message body>

“I, Rony Wilson, holder of an s/b a/c - no. 567720192, have subscribed to the e-statement facilities of SBCS bank through your e-banking portal, in May. Now it’s July, and I have not received any kind of statements – neither paper nor e-statement, for both May and June. Do check up your database and send in the statements immediately, in this email ID – that is given in the subscription request also.”
This sentence can be broken down in form of an equivalent of an LR parse tree, as shown in figure 12.

Figure 12: A basic parse tree based on the example e-mail message header

The semi-structured knowledge was embedded in the following lines of the given email:

<subject or header>

“Monthly e-statement for s/b a/c no. 567720192 not received in June 2009”

</subject or /header>

<Relationship no. or account no.> s/b a/c no. 567720192: Rony Wilson< / Relationship no. or account no.>

The parse tree in figure 12 shows the attributes specific to the input sentence [ <subject or header> “Monthly e-statement for s/b a/c no. 567720192 not received in June 2009” ] with the specific product-ID validated by input string

[<Relationship no. or account no.> s/b a/c no. 567720192: Rony Wilson < / Relationship
no. or account no.], along with the type of the attribute given in italics below the attribute labels. For interpreting these attributes, a thesaurus is needed.

In the parse tree as shown above, two basic semantic rules can be identified as follows (in the form equivalent to \( b := f(c_1, c_2, \ldots, c_k) \) as explained in the previous section)

\[ b_1 := \text{about (e-statements: is-part-of(savings-account( Account no.(value), months (set of values)))}) \]

\[ b_2 := \text{not (receiving)} \]

If these rules are shown in the parse tree, the dependency graph can be viewed as in figure 13:

**Figure 13: Semantic rules, evaluation order**

From the topological sort of a dependency graph, an evaluation order for semantic rules can be obtained. For example: the evaluation order as found from a simple topological
sort on the graph above is : b1 -> b2. There are several methods for evaluating the semantic rules also, which primarily fall into the following categories:

- Parse-tree methods: They obtain an evaluation order from a topological sort of the dependency graph constructed for each input, at compile time. The limitation of these methods is the fact that they fail to find an evaluation order if the dependency graph has a cycle.

- Rule-based methods: Here, at compiler construction time, the semantic rules associated with productions are analyzed. Therefore, for each production, the order in which the attributes are evaluated is pre-determined at compiler construction time itself.

- Oblivious methods: Here, an evaluation order is chosen without considering the semantic rules.

For the concept proposed in this thesis, the second approach i.e. rule-based methods find more suitability of applications because firstly, the approach is based on the use of a thesaurus which can store the pre-defined meaning of various phrases like about (as a descriptor), not (as the first level classifier i.e. the basic characteristic difference between a phenomenon occurring or NOT occurring). These pre-defined meanings associated to these words can be used for evaluating the semantic associations (like b1 -> b2) between the productions. Secondly, as the rule-based methods (and also the oblivious methods) do not have to explicitly construct the dependency graphs at the compile time, they are more efficient in terms of their usage of compile time and space.

Therefore, the thesaurus along with the ‘contextual meanings’ of the words stored in the thesaurus, is considered as a rule-base repository for preliminary classification of
knowledge sources. For this example, there is also a need to have access to the data-dictionary or product master information-bases about specific words / descriptors like a/c: account, s/b : savings (application-context-specific) and also the product definitions databases e.g. [{e-statements} is-part-of {s/b a/c}].

The objective in this example is to extract some knowledge elements which can be used to classify this e-mail. As the context has been explained, the classification has to be based on- 1) e-statements that is part of product ‘savings account’ for the months 05 and 06, 2) type of complaint, 3) any other information.

From the dependency graph, the evaluation order will also be stored along with the three elements as mentioned, for showing the association between the elements. For example {1) product – account no. and date,} := b1, { 2) type of complaint, 3) any other information. } := b2. Figure 13 shows the use of the thesaurus, data dictionary and product definitions in this context and the final output of the translated string in the form of a table where the basic classification information is stored.

Output of this graph would be:

b1: [ [{e-statements} is-part-of {savings account: product ID 567720192}] {time:05-06 MM}] → b2 [not {received}]}

This output can act as a message stamp which is ready to be used for preliminary classification of the messages. The whole message body of message i then can go for a full analysis using any of the unstructured text clustering schemes.
Once the dependency graph is made, transitive reduction can then be used to reduce the complexity of the available semi-structured data. For example, the email form had two semi-structured fields:

Phrase 1) <subject or header>

“Monthly e-statement for s/b a/c no. 567720192 not received in June 2009”

</subject or /header>
Phrase 2) <Relationship no. or account no.> s/b a/c no. 567720192: Rony Wilson </

Relationship no. or account no.>

Now, the dependency graph with b1 $\rightarrow$ b2 of figure 3 has been developed with only phrase 1. Using incremental computing, this dependency graph of figure 12 can be linked with phrase 2 where specific information about b1 is given. Phrase 2 is added on to the dependency graph of figure 13, as shown in figure 14.

Figure 15: Two dependency graphs based on two e-mail form-fields (Phrase 1 and phrase 2)

Following figure 14 with the inclusion of b3 for phrase 2, it can be easily deduced that dependency graph b3 is actually a sub-set of dependency graph b1$\rightarrow$b2. This relationship can be embedded in the email form field as sub-header under the main header. This relationship is shown in figure 14. When this relationship is discovered as a hierarchy between two form-fields, the common fields of b3 can be removed and only the incremental information i.e. $\Delta$[{b1$\rightarrow$b2}, b3] that is ‘Rony Wilson’ in the productID attribute can be added to b1$\rightarrow$b2.
graph itself. This way, the graph is reduced using the transitive closure of b3 within b1->b2, and the data structure Product ID gets incremented so that there is no significant information loss from both the form-fields.

The transitive closure is shown in figure 16 and the incremental data structure is shown in figure 17 which is the final output of the input example, based on the process model described in figure 11, using dependency graph and basic concepts of incremental computing.

Figure 16: Transitive closure of b3 within b1->b2
The final output information label would hence be:

[Complaint, [e-statement s/b a/c no. 567720192Rony Wilson] [05,06, MM], [not], [received]]

The elements as identified in this information label function as a pre-classifier for the main text email body, as the contents are of type ‘Complaint’, along with specific data values for product and time ID as well as functional information captured in terms- [not], [receiving].

4.4 Analysis of outcome of the process models

Advantages as shown in the generic process model and then in the example dry run, of using transitive closure on dependency-graph based pre-classifier information label generation include:
1) Reduction in data structure size and complexity: As is apparent from the example explained, the rule b3 originating from phrase 2 of the example input form-fields got absorbed by being transitively closed with b1->b2 and hence being transitively reduced to one data structure from two input form-fields.

2) Reduction in algorithmic complexity for classifier information label generation: Following the well-accepted McCabe’s cyclomatic complexity value E-N+2 (E- no. of edges, N- no. of nodes), if we evaluate the complexity of graph in figure 6 both on b3 and b1->b3 i.e. before applying transitive closure and reduction on b3 and b1->b2, we get the complexity of the data structure as 2(for b3) + 3(10-9+2 for b1->b2) = 5. The cyclomatic complexity of the transitively reduced version of the same input form-fields text as shown in figure 7 is: 9-9 + 2 = 2. Therefore, in the example, the data structure parsing complexity reduces from 5 to 2. At the same time, there is no information loss as the Product ID in figure 7 is incremented with value from b3.

Therefore, by using transitive closure-based incremental computing concepts on pre-classifier information label creation, the data structural size and complexity as well the parsing algorithmic complexity get reduced, which will speed up the pre-classifier label creation process and also make it more resource-efficient.

Possible extension of this work may include identification of two separate lexical analysis and parsing trees, one based on common language thesaurus and the other based on context/domain specific terms-base, like the ‘account, s/b’ terms as given in the specific application example in context of CRM of a bank.

This is a clean-slate approach as well as a new concept for knowledge pre-classification from unstructured sources combining incremental computing and transitive closures concepts to the concept of a preliminary classification using the tried and tested concepts of traditional
compiler construction in theoretical as well as applied computer sciences domain. The primary advantage of the approach is the fact that it can present a basic identifier or classifier for an unstructured knowledge source by exploiting some amount of structured string-type information that are usually present in the source headers or document labels or message subjects/headings. This way it can reduce the workload of a knowledge extraction module which can then take the entire body-text of the document/ message/ knowledge source and apply the well-researched approaches of unstructured text handling on them. This way the entire process of knowledge extraction becomes faster and more resource-efficient. Further research possibilities include applications and extensions of unstructured text handling approaches using other theories in the concepts of compilers, theory of computer science, theory of languages like regular grammar and CFL etc.

4.5 Validation of proposed models using Case Based Research Methodology: Knowledge pre-classifiers application in Customer Relationship Management at Domino’s Pizza

In the service industry, across the globe, the universally accepted mantras are ‘Customer is God’, and hence ‘Know your customers better’. IT and specifically, KMS (Knowledge Management Systems) can play a vital role in this process of knowing the customers- be it in using analytics for customer profiling in retail, or creditworthiness analysis in retail banking- loans verticals, or designing loyalty-based programs or promotional programs across various retail/ financial services/ hospitality industries etc. Most of the successful services organizations of today are doing at least some of these, to survive in the highly competitive marketplace.
In this context, a KMS attached to a CRM in the back-end and the POS system i.e. the front-end, for retail sector with mostly repeat customers, can bring immense business value to the organizations. This idea is modeled in this thesis, using the knowledge pre-classifier models as presented in sections 5.3 and validated with the Domino’s Pizza as an example case from the retail sector with repeating customer-base.

IT and specifically, KMS (Knowledge Management Systems) can play a vital role in this process of knowing the customers- be it in using analytics for customer profiling in retail, or creditworthiness analysis in retail banking- loans verticals, or designing loyalty-based programs or promotional programs across various retail/ financial services/ hospitality industries etc. Most of the successful services organizations of today are doing at least some of these, to survive in the highly competitive marketplace.

Knowledge-bases about customers have got two aspects: one- their personal/ professional/ other information that comes from or forms part of the customer master, two – their transactional history. The basic nature of these two kinds of data being very different, i.e. the master data being fairly static in nature and the transactional data being highly volatile and voluminous too, poses a significant challenge to knowledge management professionals when there is a need to combine both, in order to ‘know the customers better’.

This challenge is addressed in this case-based model application and validation in this thesis, using a small use-case of Domino’s Pizza, that records the customer’s personal data, but till now, has not made an effort to combine or connect that to the customers’ transactional data.

In this specific example, many customers are generally repetitive, a fact that is already being exploited in the operational model o Domino’s in which, on receiving any call for an order,
they first ask the customer his/her contact number, from which the system can retrieve other relevant details, if the customer is a repeat customer.

But, this very fact that the customers are generally repeat customers, and thus leave a significant transactional data footprint in the transactional systems of Domino’s, is not exploited in terms of the information value of these historical transactional records. For example, these historical transactional records can be analyzed and combined with the personal data, on the basis of which, a pro-active call can go from Domino’s to a high-value (this can be identified by combining all the previous transactional volumes) customer on his/ her birthday or anniversary day etc., offering him/ her a discounted/ promotional scheme for a bulk order. This way, combining the transactional data with customer master can generate more business value to the organization.

4.6 CRM and KMS- need for integration

CRM and knowledge management (KM) are still apparently seen as different disciplines, with the two sharing little, perhaps except the same server hardware and underlying data warehouse or multi-dimensional data architecture. The basic fact is often underestimated in terms of both potential and usability is that both these systems effort to improve business efficiency and customer satisfaction, to deliver continuous improvement to business clients. Subsequently, it is for the benefit of the business that they start speaking the same language and share a unified single version of truth in terms of customer data!

Knowledge-centered CRM firms like RightNow Technologies [5] have brought greater attention to the crossover, as have such mergers as ATG/Primus. KM focuses largely on finding the right solution to a problem that requires detailed insight. In this context, insight into the individual customer’s preferences and behavioral patterns- i.e. beyond the static personal data to a value-added analysis of each one’s transactional footprint histories,[6] can
bring in great value to the organization by ensuring individualized attention and thus very high customer satisfaction and delight.

Unfortunate as it may seem, many companies still have not attained the level of deep integration that ties knowledge base activity at the operational back-end, to a CRM-facing customer record or POS-based front-end. Some companies like computer peripheral manufacturer Adaptec use the intersection of CRM and KM to guide product and service decisions and attempt to waylay customer service overloads before they begin.

However, as the global economy has become more interconnected and “flat,” the pressure to increase performance has accelerated. The demands for new market share, new revenues, new levels of efficiency and new levels of accountability have become an urgent reality. In the middle of this new business reality are customers. They have more information at their fingertips, and more choice than ever before. Competitors have more access to their attention and greater ability to be ready at a moment’s notice to provide service from anywhere on the planet.

Losing a customer’s loyalty only takes an instant. And unfortunately, it happens more than most organizations would like to admit. In the customer-driven age there are no secrets. Information is plentiful and easily accessible online, and customers can learn more about an organization or a product in one day than previously was possible in a year. Consequently, while facing the customers- across whichever channels and mode of interactions, most organizations intend to put their best foot forward. They understand how important it is to present the very best the organization has to offer to their customers. And in many cases, this happens. However, when organizations fail to meet or exceed customer expectations, alternatives are only a Web search away.
In the customer-drive age, however, even the best organizations can falter and the stakes are high. Customers can easily discern when a business engages in poor service behaviors, including:

- **Inconsistency**: When a customer’s experience varies significantly across various interactions, even if his/her ideas about service quality may not suffer, the idea about a consistent and hence, stable response, suffers. In case of repeat customers, this situation can be a ‘losing grounds’ situation for any organization.

- **“Spin”**: With the wealth of information now available on the Web, customers are wary of being “sold.” In many cases, customers now know more about the products than the sales persons themselves!

- **Poor memory**: Customers expect to be remembered. No matter what an organization provides – be it a mobile phone, a consulting service or even a government benefit – customers expect their information to be current and consistent across any access point.

This third point is what is taken forward specifically in this thesis. When the customers inquire about additional products or services, or come back for further transactions/interactions/ negotiations, they expect their profiles and presences to be known – for example:

1. What they’ve purchased and when,
2. how much they paid
3. what discount they received,
4. how they prefer delivery and
5. how they prefer to shop.
Any information the customer ‘thinks’ that they had provided earlier—either directly or indirectly (i.e. through previous transactions or interactions)—they expect those all information elements to be immediately referenced, understood and used in their next transaction. Not being able to do so can often be the trigger for poor customer satisfaction, resulting in low loyalty and high churn.

### 4.6.1 The Integration Model for CRM and KMS

The model for integrating KMS and Customer profiling tool and POS Front-end, through a knowledge pre-classifier based architecture (companies like Terradata already have products to support the warehousing operationally), is presented in figure 18 as given below:

This model is now validated and explained using the Domino’s case as an example.

**The Domino’s Pizza use-case**

Domino's has developed a simple business model focused on its core strength of delivering quality pizza in a timely manner. This business model includes a delivery-oriented store design with low capital requirements, a focused menu of pizza and complementary side items, committed owner-operator franchisees and a vertically-integrated distribution system. Its earnings are driven largely from retail sales at its franchise stores, which generate royalty payments and distribution revenues to it.
For connecting to the external world of customers and demands, the company started exploring ways through which they can get orders in from the Internet, for example, and payments settlements through debit or credit cards etc., i.e. electronic channel for order collection, supply, and payment collection too- i.e. the full circle of the core business process.

The new network and system enabled enhancements in day-to-day store management through daily transmission and analysis of critical retail point-of-sale data. Each day, sales
and service data got transmitted from the individual stores to a central database for analysis. This trend data then was fed back to and used by store managers to proactively direct and improve key aspects of their store operations, from response and delivery intervals to staffing, cash flow management and sales promotions. Such timely feedback also allowed managers to effectively address and troubleshoot specific customer service issues, such as delayed order delivery. The data analysis included information on pizza sales, allowing corporate management to closely track customer sales by item size and type, which in turn helped them more effectively project sales and manage inventories.

Figure 19 shows the model qualified by the Domino’s Pizza use-case as depicted above. Domino’s, as is already apparent from the short use-case, is a highly tech-savvy organization, but it has by far used CRM primarily for operational efficiency improvements. It has not exploited it’s available technology and knowledge resources for improving customer experience by trying to offer more customized or individualized services. This model shows specific customer profile data elements that can be linked in real-time, to a customer’s past preferences and transactional histories, during his current and future interactions, thus bringing in better value propositions for him/her as well as the organization.
Figure 19: The proposed CRM-KMS-POS Integration Model explained with Domino’s use-case
The Domino’s use-case provides a highly relevant validation platform for this model, as:

1) it belongs to the retail sector, that depends most heavily on B2C Customer interactions

2) it belongs to the service sector – specifically in terms of offering food- that is highly ‘individualizable’ (a new phrase coined that’s highly appropriate to the context) in terms of tastes and preferences

The Domino’s use-case example clearly brings out what value this integrated model can offer- to the customers, and hence to the organization. Just as suppliers, customers are also nowadays viewed as internal partners/ stakeholders or value-contributors in the value-chain, rather than being just value-consumers. This is more relevant in context of repeat customers, who, by way of their loyalty, bring great insights and value to any organization, in terms of their transactional footprints- which, if analyzed adequately, can generate highly viable new product/ service ideas.

This work can be extended both in terms of the front-end i.e. the POS support side as well as in the back-end i.e. the customer data dimensions-side.

**Sample data feed examples**

The algorithms used for this is based on models given in sections 4.3 – the thesaurus based keyword identifier approach.

A Javascript-based prototype of the knowledge pre-classifier has been created as part of this research work, as shown below:
Figure 20a: JavaScript Prototype – Runtime Screenshot 1- CRM supported by customer knowledge pre-classifier from unstructured communication (developed and run by Author)

Now, following are customer interaction samples, that are fed into this form, and the initial pre-classifier information e.g. preferences for spicy/ non-spicy/ medium spicy, veg/ non-veg, allergy – mushroom, spinach etc. are identified through this pre-classifier prototype.

Customer data feed 1: “I want chicken pizza, medium spicy for a children party. It is a birthday celebration. Chicken should be light.”
Figure 20b: JavaScript Prototype – Runtime Screenshot 2- CRM supported by customer knowledge pre-classifier from unstructured communication (developed and run by Author)
The output shows the pre-classifier identified for this input:

Customer data feed2: “We would like to order for spicy pizzas. But we have allergy-mushroom, and no-mutton please. This is on guest-demand.”

This input identifies the following pre-classifier- [spicy, allergy-mushroom, no-mutton, guest-demand] as shown in the screenshot below:
Customer data feed3: “We want to order for a birthday party. Some flavors should be hot and spicy. Nothing should contain spinach. Chicken is ok.” This input identifies the following pre-classifier- [spicy, chicken, hot, birthday, party] as shown in the screenshot below:
Figure 20e: JavaScript Prototype – Runtime Screenshot 5- CRM supported by customer knowledge pre-classifier from unstructured communication (developed and run by Author)

Customer data feed4: “we have a party. pizzas will do. hot and spicy please. chicken. some have allergy-mushrooms, so no mushrooms please.” This input identifies the following pre-classifier- [spicy, chicken, allergy-mushroom, hot, party] as shown in the screenshot below:

Figure 20f: JavaScript Prototype – Runtime Screenshot 6- CRM supported by customer knowledge pre-classifier from unstructured communication (developed and run by Author)
Customer data feed5: “family party. non-spicy. chicken. do you have cold drinks? would also like to have cakes and waffles.” This input identifies the following pre-classifier- [spicy, chicken, party, non-spicy, cold drinks, cake, waffles, family] as shown below:

![Figure 20g: JavaScript Prototype – Runtime Screenshot 7- CRM supported by customer knowledge pre-classifier from unstructured communication (developed and run by Author)](image)

4.7 Analyst of Results on Pre-classifier Application in CRM at Domino’s Pizza Case

From the five example data feeds and their output classifiers as shown above, the value of this model is clearly demonstrated as it first runs every customer interaction instances/records as customer-profile knowledge inputs through the pre-classifier.

As demonstrated with the short-text customer interaction examples, knowledge pre-processing can make knowledge extraction processes faster and more resource-efficient. The basic functions similar to a pre-compiler can be used as a pre-processing unit, as analogous to the Oracle- pro*C kind of combinations. In case of such pre-compilers like Pro*C with oracle, we see that the pre-compiler primarily acts as a filter and sends the classified inputs
to different processing units or modules like a separate C compiler for processing the C programming sections and an SQL compiler for processing the ‘exec SQL …’ statements. Similarly, if this concept gets applied in pre-processing knowledge elements for creating re-usable knowledge repositories which can store integrated knowledge elements across various sources, types and structures, the knowledge extraction, capture, conversion/translation (to the format acceptable to the repository) etc. i.e. the later steps become easier and faster.

In fact, many of the knowledge elements which are generally viewed as ‘unstructured’ or ‘free-flowing texts’ have some degree of explicit structured information for example embedded in their labels. For example, the full customer interaction script may require much more resource to classify each interaction instance into a specific category. But if we have only the short texts and we use them first to create the initial classifier labels and then take the entire script for more detailed analysis, the process becomes faster and more resource-efficient. Generally, these already embedded ‘semi-structured’ information available as short-texts which can help any extraction module to do some ‘level 0’ or ‘pre-classification’, do not get adequately exploited if the whole document along with the semi-structured part also is input at the beginning itself to the extraction modules. For example, there can be limited amount of ‘pre-classification’ information embedded or available in the document headers, message headings, subject lines of letters or emails and so on. These, if adequately processed by a knowledge preprocessor before entering into the actual extraction phase, some classification information can already be made available through this pre-processing, to the extraction modules.

Therefore, the benefits of a knowledge pre-processing unit to be placed before the actual knowledge extraction and capture modules can be explained as follows:
- It can help the knowledge extraction modules, which are often extremely resource-hungry and slow (due to less availability of such computational resources), more efficient. The knowledge extraction modules tend to become slow because of their unavoidable and extreme logical and processing complexities. A pre-processed input can make the logic simpler to some extent.

- It also helps the knowledge extraction modules to exploit some amount of structured information that remained embedded in part of unstructured documents like headings etc.

The benefits are demonstrated in terms of the following:

1. The pre-classifier model identifies the relevant labels from these interaction records and creates a pre-classifier ‘tag’ for each message. The customers are identified by their phone numbers or IDs, so against each ID or phone no.s, these records along with these pre-classifier tags or labels are stored.

2. The labels help initial classification of customers preferences i.e. by seeing the labels it can be seen who prefers chicken (by frequency of occurrence in pre-classifier tags), who has mushroom allergy, who prefers spicy, who orders for parties and so on.

3. So, the customer ‘memory’ is retained in the organizational customer knowledge base of Domino’s and each time a repeat customer calls up, these tags become available and the CRM or sales POS persons can look at them and interact with the customer with these previously available knowledge and profile contexts.
4.7.1 Analysis of benefits in quantifiable terms

Using case-based methodology, the application of the prototype in context of CRM demonstrated significant qualitative benefits in terms of increased customer satisfaction by the operational CRM and sales POS team retaining customer memory.

This benefit is partially quantifiable too, in terms of the efficiency gains and performance improvement of front-line sales or operational CRM staff. These metrics improvements are also calculated in this section with regard to the example applications provided.

The average interaction time (through phone or face-to-face in a Domino’s Pizza outlet) is between 3 to 8 minutes per customer, as per inputs given by Domino’s staff during primary data collection phase for the CRM case given.

Let us take it as Avg\(_\text{response}\) = (3+8) / 2 = 5.5 minutes

This is when the customer’s profile information is NOT available to the front-line sales person (i.e. PRE-MODEL efficiency).

Reduction in this response cycle time would improve the efficiency of operational CRM in the following ways:

1) By reducing average response time, front-line sales people can serve more customers during the same time-slots. This will impact the revenue/sales per sales-person. This is a Direct Efficiency Gain.

2) By reducing response time, customers waiting time in the queue gets shortened. This leads to improved customer experience and thus return frequencies. That also translates into more sales i.e. more revenue.
4.8 POST-MODEL analysis

With the knowledge pre-classifier model being integrated with the CRM, the front-line sales person already has some knowledge about a repeat customer when s/he calls or visits next. This leads to quicker interactions as the customer does not have to explain all his/her special requests and requirements and the sales person already has the key profile data on customer’s preferences e.g. whether they prefer vegetarian/ chicken/ spicy/ non-spicy/ party food/ family dinner etc.

To what extent the interaction time will shorten would primarily depend on how many special request parameters a customer’s request has.

By analyzing the example runs on customer data feeds, the following no. of keywords can be found:

Customer data feed 1: “I want chicken pizza, medium spicy for a children party. It is a birthday celebration. Chicken should be light.”
This instance contains 6 occurrences of customer preference keywords.

Customer data feed2: “We would like to order for spicy pizzas. But we have allergy-mushroom, and no-mutton please. This is on guest-demand.”

This input identifies the following pre-classifier- [spicy, allergy-mushroom, no-mutton, guest-demand] as shown in the screenshot below:
This instance contains 4 occurrences of customer preference keywords.

Customer data feed3: “We want to order for a birthday party. Some flavors should be hot and spicy. Nothing should contain spinach. Chicken is ok.” This input identifies the following pre-classifier- [spicy, chicken, hot, birthday, party] as shown in the screenshot below:
This instance contains 5 occurrences of customer preference keywords.

Customer data feed4: “we have a party. pizzas will do. hot and spicy please. chicken. some have allergy-mushrooms, so no mushrooms please.” This input identifies the following pre-classifier- [spicy, chicken, allergy-mushroom, hot, party] as shown in the screenshot below:

![Figure 21d: Prototype Results Analysis-4]

This instance contains 5 occurrences of customer preference keywords.

Customer data feed5: “family party. non-spicy. chicken. do you have cold drinks? would also like to have cakes and waffles.” This input identifies the following pre-classifier- [spicy, chicken, party, non-spicy, cold drinks, cake, waffles, family] as shown below:
This instance contains 8 occurrences of customer preference keywords.

Each interaction on preferences on an average may take 0.2 to .5 minutes. So taking the average, 0.35 minutes can be saved per preference keyword if the sales-person already has them in the existing customer profiles, as identified by the knowledge-labels that were processed in the knowledge pre-classifier model on prior interactions of these repeat customers.

The array of preference keywords in 5 example instances: [6,4,5,5,8]

So, in instance 1, time saved: $0.35 \times 6 = 2.1$ minutes

in instance 2, time saved: $0.35 \times 4 = 1.4$ minutes

in instance 3, time saved: $0.35 \times 5 = 1.75$ minutes

in instance 4, time saved: $0.35 \times 5 = 1.75$ minutes

in instance 5, time saved: $0.35 \times 8 = 2.8$ minutes
Total time saves in these 5 random customer interaction instances POST-MODEL application is \( = 2.1 + 1.4 + 1.75 + 1.75 + 2.8 = 9.8 \) minutes.

So, the average time saves \( = 9.8/5 = 1.96 \approx 2 \) minutes approximately per customer interaction.

As per Domino’s inputs during primary research, an average Domino’s outlet receives \( >200 \) customer interactions during weekends, and \( >50 \) to \( <80 \) customer interactions during weekdays, per day.

Therefore, by using the Knowledge pre-classifier model for operational CRM repeat customer profile knowledge labels, they could save \( 200 \times 2 = 400 \) minutes i.e. 6.67 hours of sales-person’s time per day during weekends. This time can be better utilized in

- Serving more customers
- Getting new customers by offering quicker services and shorter waiting time / queues
- Improving efficiency of sales persons by increased sales per person per day.

The average reduction of 2 minutes in an average interaction time of 5.5 minutes show an efficiency gain of 44% for operational CRM front-line sales-force, in case of Domino’s pizza using the knowledge pre-classifier model for managing repeat customer profile knowledge more efficiently and meaningfully.

### 4.9 Main Research Contributions and Management Benefits

<table>
<thead>
<tr>
<th>No.</th>
<th>Research Contribution</th>
<th>Management benefits demonstrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Research problem solved: Knowledge classification issues in Knowledge</td>
<td>Management benefits of a robust Knowledge management system,</td>
</tr>
</tbody>
</table>
management – semi-structured contents in short texts e.g. messages, customer interactions, feedback, subject headers, email headers not adequately leveraged.

Solution: A knowledge pre-classifier model that can create ‘knowledge labels’ based on key-words from short texts as mentioned above, to be used as first-level classifiers. This can significantly reduce the KMS response time in classification exercises, thus improve KMS performance and resource utilization.

especially in domains like operations or CRM, is undoubtedly very high as the world is increasing moving towards a knowledge economy and organizations are competing in terms of their knowledge and knowledge management practices.

Primary challenges of KMSs have been performance-related. Especially when it comes to unstructured data as the potential knowledge sources, getting reusable knowledge and classifying the knowledge sources by their basic knowledge content remains a challenge as it becomes extremely resource-intensive and results into slow, economically unjustifiable performance of the KMSs.

The solution of a ‘pre-classifier’ and creation of a ‘knowledge label’ as given in this section addresses this KMS challenge successfully.

The structure and model of the pre-classifier has been demonstrated in the first section here with an example of a Customer complaint handling system for a consumer goods company (the microwave example).

<p>| 2 | KMS performance improved further by using the dependency graph based approach applied in the design of the | The KMS performance challenge is further addressed using the dependency graph-based approaches and |</p>
<table>
<thead>
<tr>
<th></th>
<th>basic knowledge pre-classifier model demonstrated with the example of CRM operations systems in a retail banking environment.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Application of incremental computing concepts has been combined with pre-classifier model, to reduce the volumes of ‘knowledge labels’ by making them more specific and non-repetitive, using the concept of transitive closures. The pre-classifier knowledge labels become more specific, and do not keep repeating similar or transitively linked key-words, thus increasing the ‘knowledge value’ or knowledge content of the pre-classifier labels. This is also demonstrated in the context of CRM operations systems in a retail banking environment, using the savings account statement related complaints of a customer, as example.</td>
</tr>
<tr>
<td>4</td>
<td>Building the pre-classifier based prototype based on context-specific keywords, demonstrating the prototype using the Case-based research methodology for validation. Domino’s CRM has been used here as the case. Several test cases using customer interaction records have been run and the results analyzed in terms of the value of the knowledge pre-classifier labels identified. The prototype of the model has been built in Javascript and has been tested with the CRM application in Domino’s Pizza. Using this case-based research methodology, it has been demonstrated that highly valuable repeat customer profile information can be extracted and used for 1) pre-classification of the customers based on their preferences i.e. veg/ non-veg, spicy/ non-spicy, party orders/ family orders etc., 2) the frontline sales people can benefit a great deal from these ‘pre-classifier knowledge labels’ on each customer, by retrieving these along with the customer profiles and addressing the repeat customers with their preferences and past interaction histories.</td>
</tr>
</tbody>
</table>
as identified in the profiles
These benefits extend the CRM from being a back-end operational system to a true value-add for customer interactions. As a result, customers feel happier that Domino’s keep track of their part interactions and preferences, and the frontline sales can put up a more knowledgeable sales front end for the customers to interact with.

The model has been first demonstrated using customer email messages in the following example contexts

1. Customer operations in a bank, (the savings account example)
2. Customer complaint handling system for a consumer goods company (the microwave example)
3. Complete case-based demonstration of a prototype built by the author, on application of the knowledge pre-classifier model in case of CRM in Domino’s pizza.
4. Quantifiable benefits of the model has been demonstrated with 5 sample customer data feeds in case of Domino’s pizza and an efficiency gain of 44% in front-line sales-force utilization and operational CRM efficiency has been demonstrated.

The contribution and qualitative management benefits are analyzed and summarized in the table given above.

4.10 Conclusion
In this chapter, knowledge pre-classifier techniques and process models have been presented. These have been validated using small case-lets from retail and banking CRM
environments first in terms of customers’ interactions with organizations as main knowledge sources. Then the complete case of Domino’s Pizza Operational CRM has been presented and the models’ prototype has been run with customer interaction instances and subsequent improvement in customer satisfaction and sales force efficiencies have been analyzed both quantitatively and qualitatively and have been summarized in a tabular form.