Major Project MidTerm report on

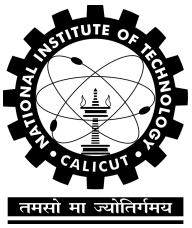
EXTRACTION OF QUESTIONS FROM THE INTERNET USING A MACHINE LEARNING APPROACH

Done by

Alok Saw	B090924CS
Jerrin Shaji George	B090437CS
Shubhangam Agrawal	B090904CS
Stein Astor Fernandez	B090006CS

Guided by

Dr. Priya Chandran



Department of Computer Science and Engineering

National Institute of Technology Calicut

Calicut, Kerala 673 601

Winter Semester 2013

Abstract

We propose a Support Vector Machine based approach to extract questions pertaining to a topic from the internet. We adapt certain features from SQUINT[1] to identify relevance of text and propose 2 new features to identify questions in a text section.

This Machine Learning component is being developed as an application which takes a topic as input and produces a list of questions related to the input topic as output.

Contents

Chapter 1

Introduction

Today, there is a massive amount of data available on the internet. While advancements in search technologies have made searching for relevant pages trivial, each page contains a lot of irrelevant data, images and advertisements. Thus, having some way to extract only the relevant sections of data from the huge set of search results would result in massive effort and time savings.

There is often a need for students as well as faculty to obtain a list of questions related to certain topics while studying or teaching these topics. The internet has a vast number of such questions for any such topic, however one must know what terms to search for and then manually visit each page returned in the search results and find out such questions in a tedious manner.

We plan to make an application to automate this process which will take a topic as input and return a list of questions based on the topic from the internet.

1.1 Problem Statement

Given input a topic $\mathbf{t}$, search the internet for pages which may contain questions related to $\mathbf{t}$ and output a set $\mathbf{R}$ comprising of relevant questions extracted from these pages.

Chapter 2

Literature Survey

The trivial way of approaching this problem would be to simply search for the presence of the input topic in the various sections and output matches - however this would not yield a good performance and would have no way to identify questions. Teevan et al.[2] have discussed the relevance of using search methodologies that focus more on contextual information as opposed to simple keyword matching as that may not be enough the capture the relevant text.

Thus rather than simple keyword matching, we must be able to identify patterns in the text sections which make it a question and also make it relevant to the input topic. Machine Learning is a good approach to solve this problem as it finds the various patterns in the text without the need for detailed statistical analysis.

Arthur Samuel defines Machine Learning as the *Field of study that gives computers the ability to learn without being explicitly programmed.*

Machine learning, a branch of artificial intelligence, is concerned with the design and development of algorithms that take as input empirical data and yield patterns or predictions thought to be features of the underlying mechanism that generated the data.

A learner can take advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distribution. Data can be seen as instances of the possible relations between observed variables.

A major focus of machine learning research is the design of algorithms that recognize complex patterns and make intelligent decisions based on input data[3].

Machine learning problems can be categorized broadly into 2 categories[4] -

Supervised Learning The goal is to predict the value of an outcome measure based on a number of input measures.

Unsupervised Learning There is no outcome measure, and the goal is to describe the associations and patterns among a set of input measures.

As our problem is to predict whether each section is a related question or not, we see that 'Supervised Learning' is the correct approach to our problem.

Joachims has also concluded[5] that Support Vector Machines are an effective text classification tool which uses the Supervised Machine Learning model. This is due to the high dimensionality of the feature vector[1].

Inoue et al.[1] propose an SVM based approach to identify the most relevant subsection in Web pages returned by a search query (*SQUINT*). They have obtained very accurate results using this model.

A Support Vector Machines take a set of input data and predicts, for each given input, which of two possible classes forms the output, making it a non-probabilistic binary linear classifier.

They construct a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier[6].

Our problem is similar to *SQUINT* with the significant challenge being the identification of questions. Thus, we explore a Support Vector Machine based approach similar to SQUINT in order to detect relevance to the input topic and aim to identify various features which indicate that a section is a question.

Chapter 3

Work Done in S7

3.1 Architecture Design

We identified a component—based architecture for the application which would be feasible to implement.

A component based architecture is suitable for this application since each component has a well defined function which is dependent only on the output of the other subsections.

This will help in having a de-coupled structure and each component can be modified without affecting the other components. We can also observe the state of data at each section for better debugging and analysis.

The high—level overview of the architecture is as follows :

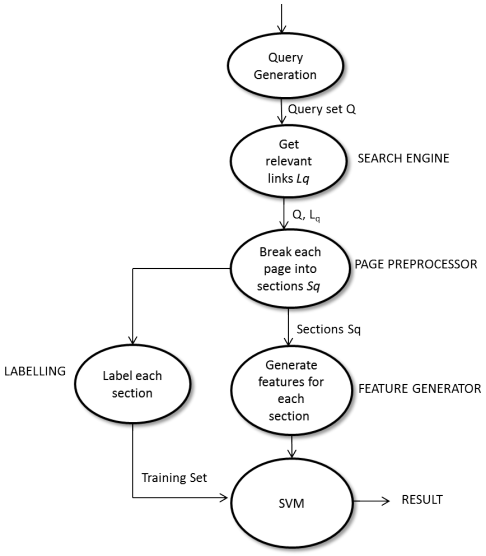


Figure 3.1: Architecture Diagram

3.1.1 Query Generation

- Take the topic **t** as input.
- Generate set **T** of related subtopics and add the topic to **T** as well.
- Query Set **Q** := **T** x {'questions', 'problems', 'solved examples'}.

Firstly, we will take the input of the topic and generate a list of sub-topics and other keywords which relate to the topic. For eg. if the topic is 'Operating Systems', various sub-topics can be 'Processes', 'Threads', 'Cache' etc. These will form the set **T**.

Each phrase in **T** will be appended with chosen suffixes such as *questions* and *problems* to generate the query set Q which will be queried on the search engine.

3.1.2 Search Engine

$\forall \mathbf{q} \in \mathbf{Q}$:

- Search the internet for the query phrase **q** using Google Custom Search API (or any other search engine).
- Add the top *nLinksPerQuery* links to set L_q.

3.1.3 Page Preprocessor

$\forall \mathbf{l} \in L_q$:

- Load the link **l** using Jsoup package and use the HtmlToPlainText class to strip away all markup data, images etc. and yield only the text portions.
- The text of the page is then broken up into sections on the basis of -
  - Number of lines
  - Paragraph Structure
  - Interrogative Indicators - *'what', 'how', 'explain', 'define', '?', 'elucidate'* etc.
- Each section **s** thus obtained is added to the section set S_q.

For each query phrase $\mathbf{q} \in \mathbf{Q}$, we will load each link we have in our link set L_q using Jsoup package functions and use the HtmlToPlainText class to strip it of all useless markup data, links and pictures and parse out only the text portions available on the page which may contain the questions.

Each page will then be broken into various sections. This will be done by a parser which will determine *good* sections which may be questions on the basis of a maximum number of lines, the paragraph tags of html and interrogative indicators like a line ending with a *'?'* or lines starting with words like *'what', 'define'* etc. as these are likely to be starting or ending words of a question.

Each section **s** thus obtained will be added to the section set S_q for a query.

3.1.4 Feature Generator

The feature generator will generate the features and statistics for each subsection, which will be analysed by the SVM to create the regression model.

We adapt the first three features from *SQUINT* to determine the relevance of a section to the query phrase[1].

Furthermore, we propose two additional features to determine whether a section contains a question or not.

The features are —

1. Word Rank Based Feature
2. Bigram Rank Based Feature
3. Coverage of Top Ranked Tokens
4. Coverage of Interrogative Indicators
5. Absence of Specific Keywords

Word Rank Based Features

The rank of a word is defined to be its position in the list if the words were ordered by frequency of occurrence in S_q[1].

We would have a feature each for say, the top 200 most frequent words in S_q.

For the i^{th} ranked word, this feature would basically have the value for the frequency of this word in the current section.

We can limit the dimensionality of the input vector by bucketing words by a certain range of ranks. For example, we can bucket ranks 1-5, 6-10, 11-15...etc to aggregate word counts, and come up with a feature vector of reduced dimensionality.

We also normalize for the length of the section since we do not want to be biased towards long sections.

The optimal number of words and the bucket size needs to be determined experimentally.

Bigram Rank Based Features

A bigram is defined to be two consecutive words occurring in a section[1].

This feature is computed in a manner similar to the previous set of features.

This feature is based on the intuition that the correlation between two words might be more informative than the words taken individually. For instance, 'machine learning' suggests a stronger relation to a query 'AI SVM' than the individual words 'machine' or 'learning'.

For this feature as well, we will adjust the dimensionality by bucketing and limiting coverage of ranks depending on experimental results.

Coverage of Top Ranked Tokens

Relevance to a topic may also be captured by the coverage of top ranked token types in the section[1].

For example, if we have a bucket size of 5, we might be interested in knowing how many of the top 5 ranked words occur in this section, how many of the next 5 highly ranked words occur in this section and so forth.

Specifically, if the top 5 ranked token types are 'learning', 'machine', 'data', 'access', and 'database', and a section contained 'learning' and 'data', the corresponding value for this feature is 2.

Coverage of Interrogative Indicators

Presence of words such as *'what', 'why', 'explain', 'define', 'elucidate'* etc. are strong indicators that the section contains a question.

Thus we propose this feature whose value is the coverage of a predefined set of such interrogative indicators present in the section.

We need to experimentally determine whether the frequency or coverage of interrogative indicators is a better feature.

Absence of Specific Keywords

There are certain words or patterns, the presence of which strongly indicate a section of text to not have a question.

For example, if a sentence begins with a *'Yes'* or *'No'*; or certain words like *'because'* are present, there is a high chance that the section is not a question.

We check for the coverage of such words in the section using a pre-defined list.

This feature is used as a negative feature as a higher value indicates a lower probability of the section being a question.

3.1.5 Labelling (Training Set Generation)

For generating the training set, we will manually label a set of sections as to whether they are relevant questions of the given topic or not and this data will be used to train the SVM to generate the model.

3.1.6 Support Vector Machine

We will use a Support Vector Machine with a Linear Kernel as recommended by Joachims[5].

This component will be trained using the Training Set comprising of the section sets with the generated feature vectors and manually tagged labels. The training data will be analysed to generate a regression model.

During the real runs of the application, the model generated during training will be used by the SVM to identify the relevant sections and tag them appropriately.

The sections tagged as 'relevant' will be added to the result set **R**.

3.1.7 Output

The output component will take all the sections tagged as 'relevant question' present in set **R** and output them in a clean format.

3.2 Prototype

JAVA was chosen as the programming language owing to the availability of well supported classes to implement SVMs and the object-oriented features in general.

We made a prototype program with placeholder stub functions for each component. Each stub did no useful work but just passed the data along.

Each component would be individually developed and further refined to produce the optimal output.

Chapter 4

Work Done in S8

4.1 Class Design

The overall design consists of 6 components.

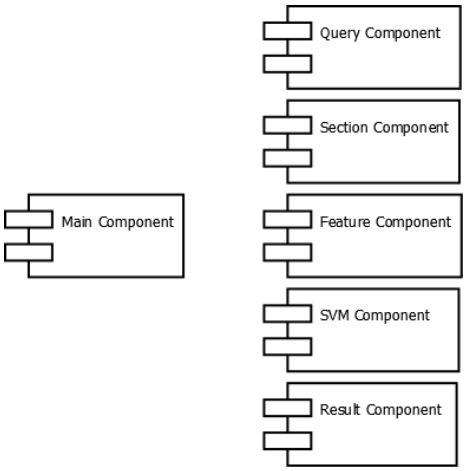


Figure 4.1: Component Diagram

4.1.1 Main Component

The main component stores the overall data and its classes provided interfaces to access and modify this data. All other components necessarily interact with the Main Component.

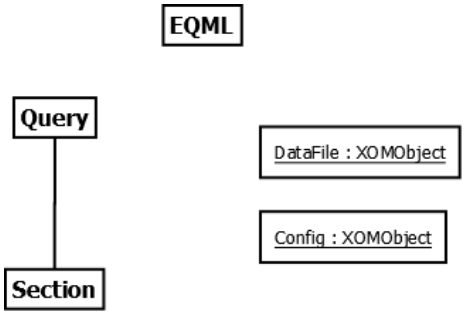


Figure 4.2: Main Component

EQML The main class which loads the other classes as required.

DataFile Interface to the data file.

Config Has all the config parameters as set in config file.

Query Stores the details about each query. Has a list of associated Section objects.

Section Stores the details about a text section and relevant attributes.

4.1.2 Query Component

The query component is responsible for taking the input topic, then generating the requisite subtopics and queries related to the same.

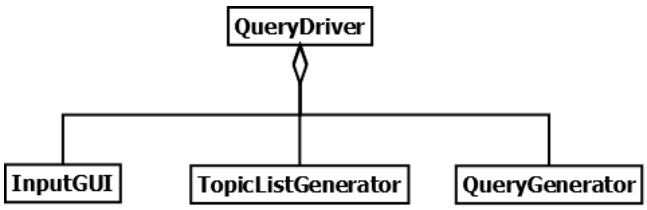


Figure 4.3: Query Component

QueryDriver Loads the rest of the classes and interfaces between them. Finally returns a list of Query objects.

InputGUI Draws the GUI Window where the user provides the input topic.

TopicListGenerator Generates a list of related and/or sub–topics related to the input.

QueryGenerator Generates list of Query objects by adding various suffixes to given topic.

4.1.3 Section Component

The section component processes each query and generates a list of Section objects for each query.

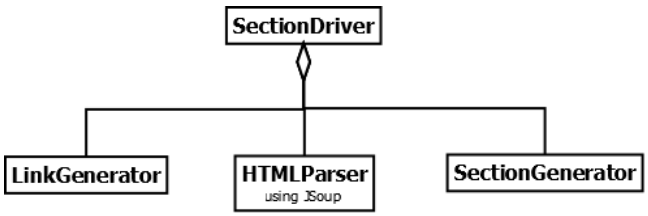


Figure 4.4: Section Component

QueryDriver Loads the rest of the classes and interface between them. Finally attaches a list of section objects to each Query object.

LinkGenerator Uses a Search API to search for the query and generate links of the top results.

HTMLParser Uses the JSoup package to load links and return plaintext form of the web–pages.

SectionGenerator Breaks pages into sections and makes an Object for each.

4.1.4 Feature Component

The feature component has classes to generate all the requisite features for the sections.

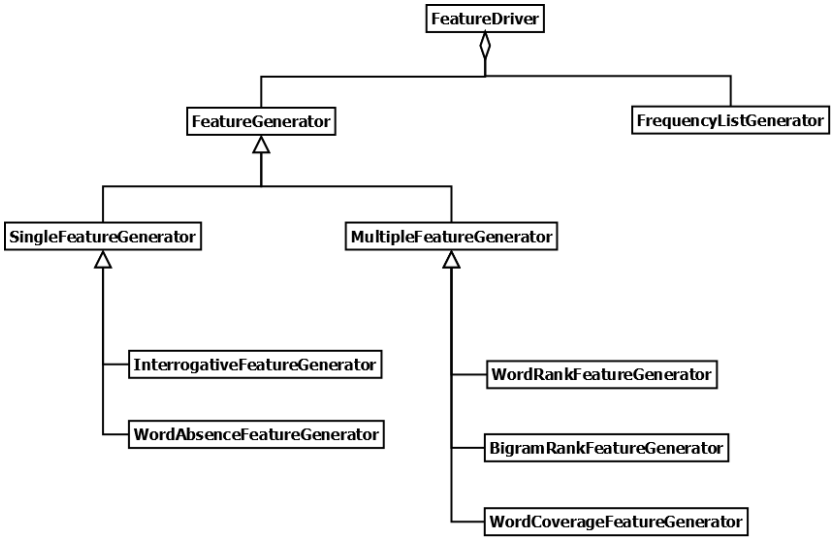


Figure 4.5: Feature Component

FeatureDriver Loads the rest of the classes and interface between them.

FrequencyListGenerator Processes all the sections related to a query to generate a list of words and bigrams in descending order of frequency.

FeatureGenerator Interface for feature generation.

SingleFeatureGenerator Interface for feature—types that generate a single feature.

MultipleFeatureGenerator Interface for feature—types that generate a list of features.

WordRankFeatureGenerator Generates the Word Rank based feature list of the section.

BigramRankFeatureGenerator Generates the Bigram Rank based feature list of the section.

WordCoverageFeatureGenerator Generates the Word Coverage based feature list of the section.

InterrogativeFeatureGenerator Generates the Interrogative indicator frequency based feature of the section.

WordAbsenceFeatureGenerator Generates the feature based on absence of specific words of the section.

4.1.5 SVM Component

Consists of all the classes which are related to the machine learning aspect of the program.

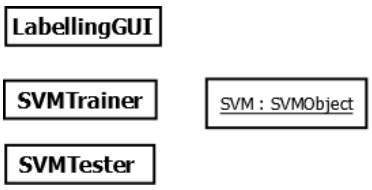


Figure 4.6: SVM Component

LabellingGUI Loads each section one by one and allows the user to manually tag the section. This is done to generate the Training and Testing sets.

SVM The object which implements the Support Vector Machine.

SVMTrainer Selects the training set and loads the SVM object with the feature vectors in order to train the SVM.

SVMTester Selects the testing set and runs the sections on SVM. Analyses the results and generates a performance report.

4.1.6 Output Component

This component is responsible for generating the list of questions as output.

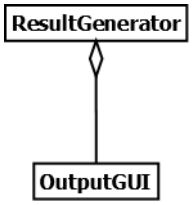


Figure 4.7: Output Component

ResultGenerator Uses the SVM component to generate tags for all the sections and calls the OutputGUI.

OutputGUI Generates a list of the sections with Positive Tags and outputs them to a GUI Window.

4.2 Data Structure Design

We have chosen to use an n–ary tree as the data structure keeping in mind the component–based architecture.

The tree structure allows us to model the data easily while being completely extensible at any point. It allows quick access to required data and allows us to group data in a functional manner.

4.2.1 Node Design

The node will have the following attributes :

Type The type of the node eg. Query, Section

Data The corresponding data eg. 'SVM Questions'

ChildList List of child nodes

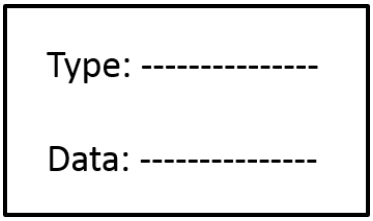


Figure 4.8: Node

4.2.2 Tree Design

The tree will look like the following :

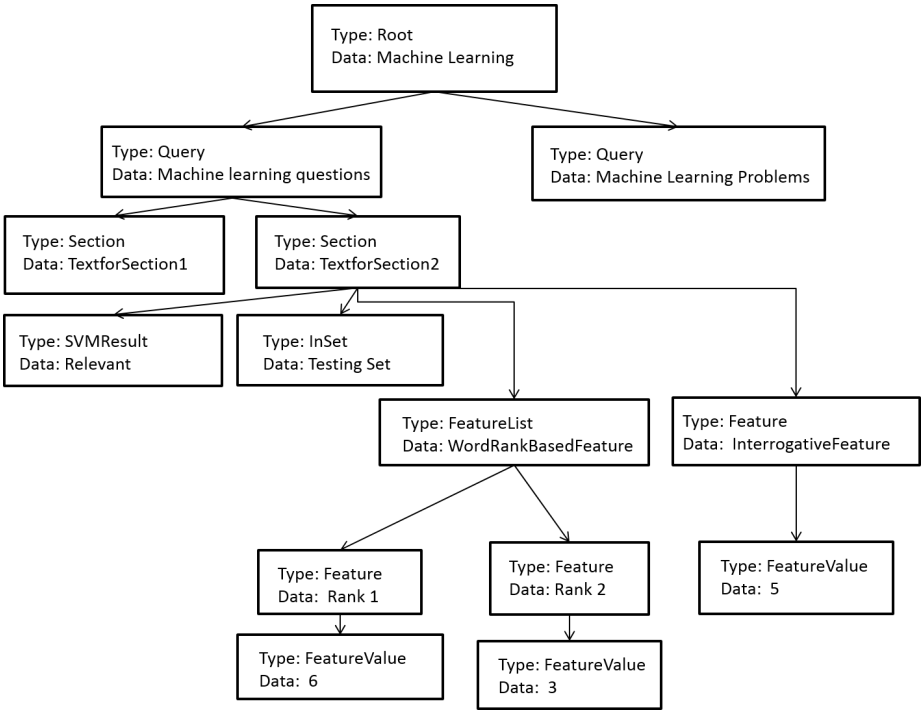


Figure 4.9: The data tree

Each node will have its corresponding data in its children.

As each component is run, it will add or delete nodes as required.

Examples of the data tree at various stages in its life–cycle.

Query Generation

The Query Generator generates a set of queries to search for from the given search topic and creates a set of corresponding nodes as children to the root node, as shown.

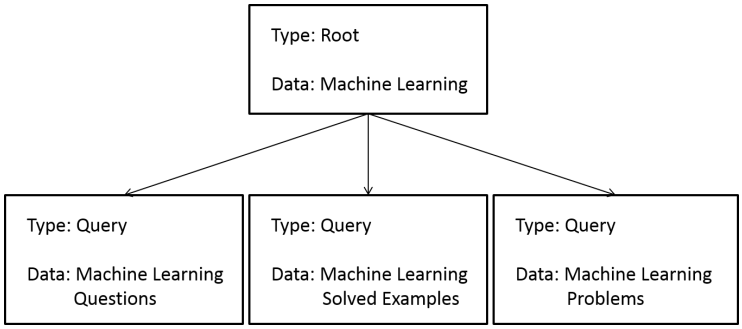


Figure 4.10: Query Generation Data

Link Generation

For each of the queries generated in the previous step, a set of links corresponding to the search results for that query is generated as children of each query node.

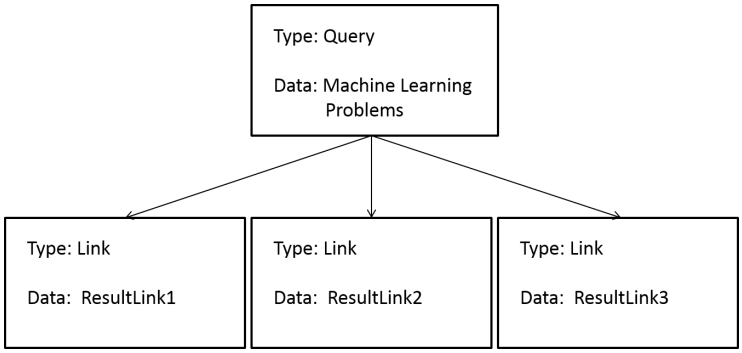


Figure 4.11: Link Generation Data

Section Generation

The Section Generator parses each of the links generated in the previous step, removing unnecessary data and then divides each such page into sections, each of which may contain a question relevant to the topic at hand.

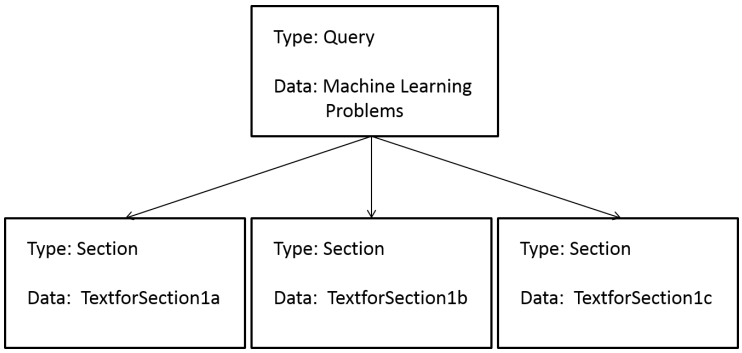


Figure 4.12: Section Generation Data

Labelling

Sections intended to be part of the training or testing set will be labelled as relevant or not relevant. This is indicated by a corresponding child node for that section node, as shown.

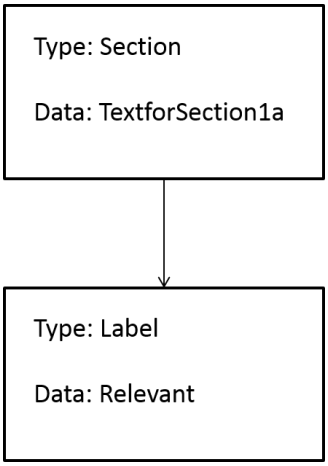


Figure 4.13: Labelling Data

Feature Generation

The features of a section are similarly represented by corresponding child nodes of each feature category.

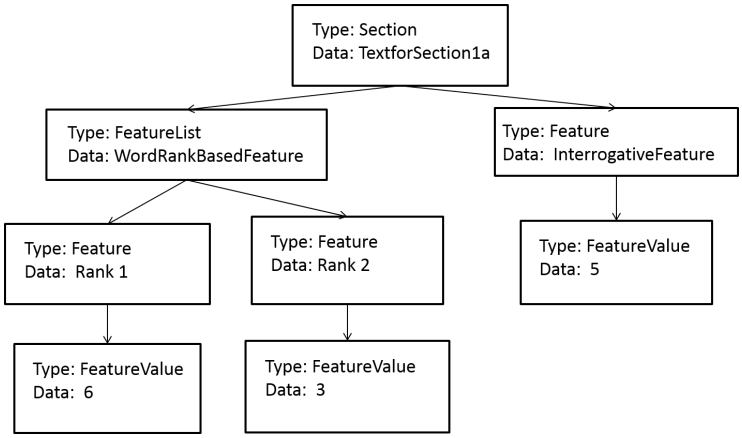


Figure 4.14: Feature Data

SVM Classification

The results of the Support Vector Machine classification and the set to which the result belongs are also represented as shown.

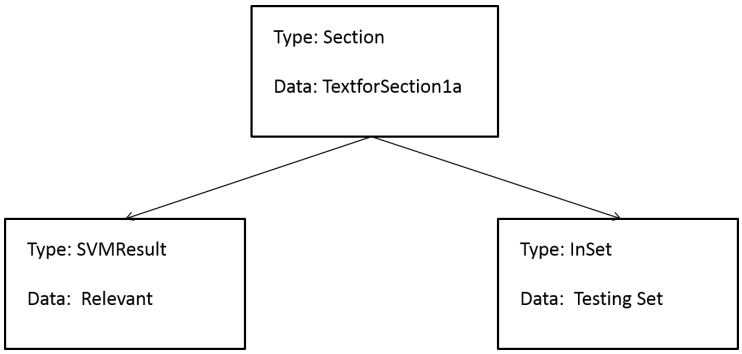


Figure 4.15: SVM Data

4.3 Data File Design

We will be using the XML file format for the data file since the tree–like structure of the XML format is highly suitable for representing a tree data structure[9].

The file, *data.xml*, will be a representation of the data stored in a tree format.

Each node of the tree will be represented as an element[1] in the XML file delineated by the tags of the node–type. Each child–node in the tree will be represented as child–elements of the corresponding parent node.

The key features of this design are its performance, extensibility and transparent nature which allows us to analyse the file at any point in the application's life-cycle.

The JAVA XOM[2] class is capable of loading the data from the file directly into a tree format. This will allow us to easily read, write or update the data file as required.

[1] A logical document component which either begins with a start-tag and ends with a matching end-tag. The characters between the start- and end-tags, if any, are the element's content and may contain other elements, which are called child elements. Eg. <Type>Data<\Type>.

[2] XOM is XML object model. It is an open source, tree-based API for processing XML with Java.

```
<root>
    Machine Learning
    <query>
        Machine Learning Questions
        <section>
            Text for Section1
            <svmresult>
                Relevant
            </svmresult>
            <inset>
                Testing Set
            </inset>
            <featurelist>
                WordRankBasedFeature
                <feature>
                    Rank 1
                    <featurevalue>6</featurevalue>
                </feature>
                <feature>
                    Rank 2
                    <featurevalue>3</featurevalue>
                </feature>
            </featurelist>
            <feature>
                InterrogativeFeature
                <featurevalue>5</featurevalue>
            </feature>
        </section>
        <section>
            Text for Section2
        </section>
    </query>
    <query>
        Machine Learning Problems
    </query>
</root>
```

Figure 4.16: data.xml example

4.4 Config File Design

The file *config.xml* will have all the configurable parameters in the XML format.

The values will be loaded from this file every time the application runs. Thus we need not re–compile the entire application just to change the parameters.

The parameters can be changed in a simple manner using any text–editor.

Thus the user is free to change the parameters as to his/her requirements. This will help us reduce wastage of time during testing and analysis of parameters by eliminating needless re-compilations.

4.5 SVM Package Identification

We have identified the LIBSVM[10] library to implement the Support Vector Machine.

We are using LIBSVM because it has functions for the Linear Kernel[11] that our algorithm proposes to use. It also implements the Polynomial Kernel and the Radial Basis Function Kernel (a non-linear kernel) which we can use to compare accuracy between the different methods.

LIBSVM is a mature SVM library for JAVA and has good performance and high accuracy[11].

4.6 Coding Progress

We have spent most of the time zero-ing in on the class design and data design so that each class can be coded to an agreed–upon interface. We have also spent time researching and selecting good libraries for our needs so that we need not waste time re–inventing the wheel and can use these stable libraries for file–access and the machine learning component.

Having clearly defined the architecture and delineated the different modules of the application, we can independently develop each module.

To this end, we have assigned ourselves sets of classes and have started to code them.

Once all of the modules have been completed, they can be integrated to form a cohesive application that can be refined and fine-tuned easily in order to provide optimal results.

Chapter 5

Further Work

While the overall architecture and design of the application seems promising, we need to experimentally determine the parameters which will produce the best results.

We will first finish coding the application and refine each component by performing iterations of training, testing and validation.

Once the application is ready, we will perform analysis with respect to

- Relevance of the features
- Weightage of the features in SVM classification
- Impact of the configurable parameters

We will generate a Training and Testing Set using the various components of our application. We will run the SVM component with these sets as input and record the performance impacts of various features and parameters.

This analysis will also help us determine the optimal parameters and we will use these values for the final application.

This research will indicate whether an SVM with our proposed features can identify relevant questions from web pages and if so, we should be able to output a list of questions on providing a topic as input.

References

[1] SQUINT - SVM for Identification of Relevant Sections in Web Pages for Web Search, **Riku Inoue, Siddharth Jonathan J.B., Jyotika Prasad**, *Department of Computer Science, Stanford University*

[2] The Perfect Search Engine is Not Enough : A Study of Orienteering Behavior in Directed Search, **Jaime Teevan, Christine Alvarado, Mark S. Ackerman and David R. Karger**, *Proceedings of the SIGCHI conference on Human factors in computing systems. pp. 415-422, April 2004.*

[3] Wikipedia article on Machine Learning, http://en.wikipedia.org/wiki/Machine_learning

[4] The Elements of Statistical Learning: Data Mining, Inference, and Prediction, **Trevor Hastie, Robert Tibshirani, Jerome Friedman**

[5] Text Categorization with Support Vector Machines: Learning with Many Relevant Features, **Thorsten Joachims**, *Universitat Dortmund, Informatik LS8, Baroper Str. 301, 44421 Dortmund, Germany*

[6] Wikipedia article on Support Vector Machine, http://en.wikipedia.org/wiki/Support_vector_machine

[7] Machine Learning, http://en.wikipedia.org/wiki/Machine_Learning

[8] Machine Learning Course on Coursera, https://class.coursera.org/ml-2012-002/class/index

[9] XML, http://en.wikipedia.org/wiki/XML

[10] LIBSVM, http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[11] LIBSVM, **Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin**, April 15, 2010.