PROJECT REPORT

MINING ONLINE REVIEWS FOR PRIDICTING SALES PERFORMANCE

Department of Computer Science & Engineering

MAY 2014

Under the Supervision of

Dr. Rajni Mohana

By

Kritesh Garg(101221)

Submitted in partial fulfillment of the requirements

For the degree of

BACHELOR OF TECHNOLOGY

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

WAKNAGHAT

Table of Content

Table of Figure:

CERTIFICATE

This is to certify that the work titled "**MINING ONLINE REVIEW FOR PRIDICTING SALES PERFORMANCE**" submitted by kritesh garg in partial fulfillment for the award of degree of B.Tech of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor : ………………..

Name of Supervisor : Dr. Rajni Mohana

Designation :

Date:

ACKNOWLEDGEMENT

I would like to express our sincere thanks and gratitude to our guide **Dr.Rajni Mohana** for his valuable guidance and suggestions. I am highly indebted to him for providing me an excellent opportunity to learn and present my studies in the form of this project.

I take this opportunity to thank the members of the teaching and non-teaching staff of JUIT for the timely help extended by them.

Lastly thanking our parents, for their morale support and encouragement.

Signature of the student : ………………….

Name of Student: kritesh garg

Date:

CHAPTER 1: Introduction

People’s opinion has become one of the extremely important sources for various services in ever-growing popular social networks. In particular, online opinions have turned into a kind of virtual currency for businesses looking to market their products, identify new opportunities, and manage their reputations. Hence, a mechanism that can provide users with condensed descriptions of documents will facilitate the delivery of digital content. This paper explores and designs a mobile system for movie rating and review summarization in which semantic orientation of comments, the limitation of small display capability of cellular devices, and system response time are considered. Practically, when we are not familiar with a specific product, we ask our trusted sources to recommend one.

Today, the popularity of the Internet drives people to search for other people’s opinions from the Internet before purchasing a product or seeing a movie. Many websites provide user rating and commenting services, and these reviews could reflect users’ opinions about a product. For example, the customer-review section in Amazon.com lists the number of reviews, the percentage for different ratings, and comments from reviewers. When people want to purchase books, CDs, or DVDs, these comments and ratings usually influence their purchasing behaviors. In addition to these websites, a search engine is another important source for people to search for other people’s opinions. When a user enters a query into a search engine, the search engine examines its index and provides a listing of best-matching web pages according to its criteria, usually with a short summary containing the document’s title and, sometimes, parts of the text.

1.1: AIM

This aim of this project is to develop software that will be able to read the blog or tweet from the web and then it will tell us the reviews based on the sentiments. For this we will approach to design a system that minimizes the ambiguity and redundancy problems. so that people can find out the review of the particular product or the movie etc.

We will try to implement a stemming algorithm, a classifier and a naïve bias to fulfill our project requirement.

1.2:EXISTING SYSTEM

A probabilistic latent semantic analysis (plsa) approach has already been presented that calculate the words and then provides the reviews on the basis of calculation. Plsa is a mathematical approach towards the calculation of total words. In plsa it searches for the total words and then calculates for the sentiment of the words and then provide the overall positivity or negativity.

1.3: DRAWBACKS OF THE EXISTNG SYSTEM

- Does not convert the word into root word.
- Redundancy is not removed.
- Cannot remove the repeated words.
- Cannot identify the sentiments.

CHAPTER 2: TECHNICAL DETAILS

Software Environment

Java Technology

Java technology is both a programming language and a platform.

The Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called *Java byte codes* —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following fig. illustrates how this works.

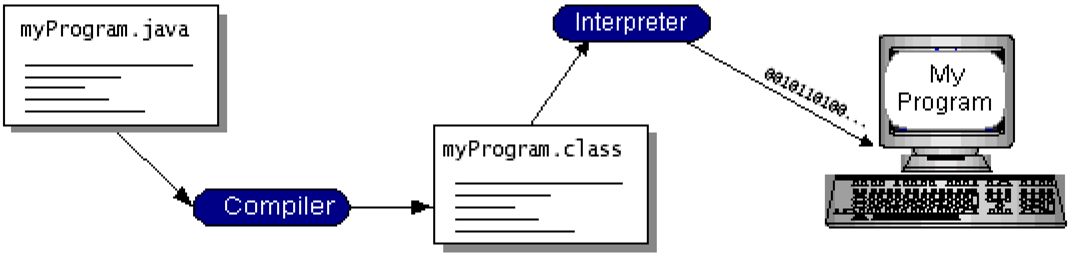
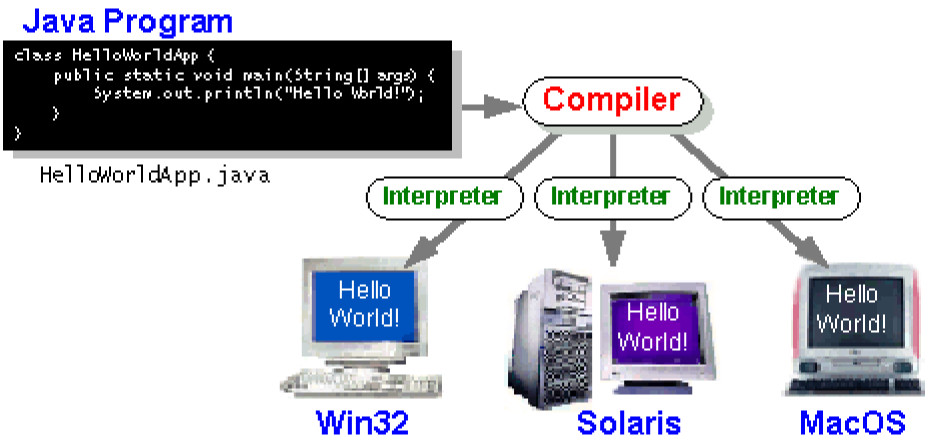


Fig. 1: java environment

You can think of Java byte codes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make "write once, run anywhere" possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.



```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Fig. 2:compiling and processing

The Java Platform

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, What Can Java Technology Do? Highlights what functionality some of the packages in the Java API provide.

The following fig. depicts a program that's running on the Java platform. As the fig. shows, the Java API and the virtual machine insulate the program from the hardware.

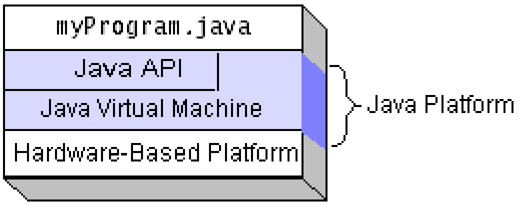


Fig. 3: java extension

Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

What Can Java Technology Do?

The most common types of programs written in the Java programming language are *applets* and *applications*. If you've surfed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a *servlet*. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- **The essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets**: The set of conventions used by applets.
- **Networking**: URLs, TCP (Transmission Control Protocol), UDP (User Data gram Protocol) sockets, and IP (Internet Protocol) addresses.
- **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- **Security**: Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- **Software components**: Known as JavaBeans^{TM}, can plug into existing component architectures.
- **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- **Java Database Connectivity (JDBC^{TM})**: Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following fig. depicts what is included in the Java 2 SDK.

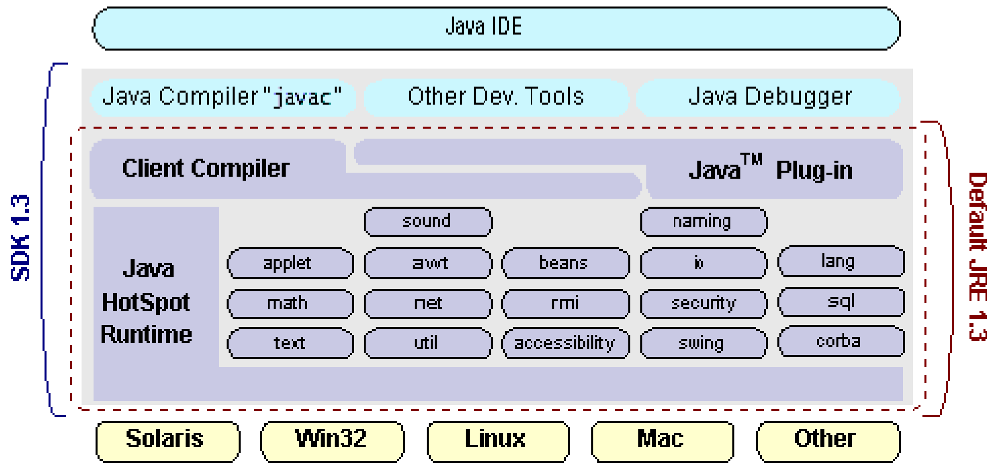


Fig. 4: java database connectivity

How Will Java Technology Change My Life?

We can’t promise you fame, fortune, or even a job if you learn the Java programming language. Still, it is likely to make your programs better and requires less effort than other languages. We believe that Java technology will help you do the following:

- **Get started quickly**: Although the Java programming language is a powerful object-oriented language, it’s easy to learn, especially for programmers already familiar with C or C++.
- **Write less code**: Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.
- **Write better code**: The Java programming language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people’s tested code and introduce fewer bugs.
- **Develop programs more quickly**: Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code and it is a simpler programming language than C++.
- **Avoid platform dependencies with 100% Pure Java**: You can keep your program portable by avoiding the use of libraries written in other languages. The 100% Pure Java^{TM} Product Certification Program has a repository of historical process manuals, white papers, brochures, and similar materials online.
- **Write once, run anywhere**: Because 100% Pure Java programs are compiled into machine-independent byte codes, they run consistently on any Java platform.
- **Distribute software more easily**: You can upgrade applets easily from a central server. Applets take advantage of the feature of allowing new classes to be loaded “on the fly,” without recompiling the entire program.

ODBC

Microsoft Open Database Connectivity (ODBC) is a standard programming interface for application developers and database systems providers. Before ODBC became a *de facto* standard for Windows programs to interface with database systems, programmers had to use proprietary languages for each database they wanted to connect to. Now, ODBC has made the choice of the database system almost irrelevant from a coding perspective, which is as it should be. Application developers have much more important things to worry about than the syntax that is needed to port their program from one database to another when business needs suddenly change.

Through the ODBC Administrator in Control Panel, you can specify the particular database that is associated with a data source that an ODBC application program is written to use. Think of an ODBC data source as a door with a name on it. Each door will lead you to a particular database. For example, the data source named Sales Fig.s might be a SQL Server database, whereas the Accounts Payable data source could refer to an Access database. The physical database referred to by a data source can reside anywhere on the LAN.

The ODBC system files are not installed on your system by Windows 95. Rather, they are installed when you setup a separate database application, such as SQL Server Client or Visual Basic 4.0. When the ODBC icon is installed in Control Panel, it uses a file called ODBCINST.DLL. It is also possible to administer your ODBC data sources through a stand-alone program called ODBCADM.EXE. There is a 16-bit and a 32-bit version of this program and each maintains a separate list of ODBC data sources.

From a programming perspective, the beauty of ODBC is that the application can be written to use the same set of function calls to interface with any data source, regardless of the database vendor. The source code of the application doesn't change whether it talks to Oracle or SQL Server. We only mention these two as an example. There are ODBC drivers available for several dozen popular database systems. Even Excel spreadsheets and plain text files can be turned into data sources. The operating system uses the Registry information written by ODBC Administrator to determine which low-level ODBC drivers are needed to talk to the data source (such as the interface to Oracle or SQL Server). The loading of the ODBC drivers is transparent to the ODBC application program. In a client/server environment, the ODBC API even handles many of the network issues for the application programmer.

The advantages of this scheme are so numerous that you are probably thinking there must be some catch. The only disadvantage of ODBC is that it isn't as efficient as talking directly to the native database interface. ODBC has had many detractors make the charge that it is too slow. Microsoft has always claimed that the critical factor in performance is the quality of the driver software that is used. In our humble opinion, this is true. The availability of good ODBC drivers has improved a great deal recently. And anyway, the criticism about performance is somewhat analogous to those who said that compilers would never match the speed of pure assembly language. Maybe not, but the compiler (or ODBC) gives you the opportunity to write cleaner programs, which means you finish sooner. Meanwhile, computers get faster every year.

JDBC

In an effort to set an independent database standard API for Java; Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of “plug-in” database connectivity modules, or *drivers*. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.

To gain a wider acceptance of JDBC, Sun based JDBC’s framework on ODBC. As you discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.

JDBC was announced in March of 1996. It was released for a 90 day public review that ended June 8, 1996. Because of user input, the final JDBC v1.0 specification was released soon after.

The remainder of this section will cover enough information about JDBC for you to know what it is about and how to use it effectively. This is by no means a complete overview of JDBC. That would fill an entire book.

JDBC Goals

Few software packages are designed without goals in mind. JDBC is one that, because of its many goals, drove the development of the API. These goals, in conjunction with early reviewer feedback, have finalized the JDBC class library into a solid framework for building database applications in Java.

The goals that were set for JDBC are important. They will give you some insight as to why certain classes and functionalities behave the way they do. The eight design goals for JDBC are as follows:

1. ***SQL Level API***

The designers felt that their main goal was to define a SQL interface for Java. Although not the lowest database interface level possible, it is at a low enough level for higher-level tools and APIs to be created. Conversely, it is at a high enough level for application programmers to use it confidently. Attaining this goal allows for future tool vendors to “generate” JDBC code and to hide many of JDBC’s complexities from the end user.

2. ***SQL Conformance***

SQL syntax varies as you move from database vendor to database vendor. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through it to the underlying database driver. This allows the connectivity module to handle non-standard functionality in a manner that is suitable for its users.

3. ***JDBC must be implemental on top of common database interfaces***

The JDBC SQL API must "sit" on top of other common SQL level APIs. This goal allows JDBC to use existing ODBC level drivers by the use of a software interface. This interface would translate JDBC calls to ODBC and vice versa.

4. ***Provide a Java interface that is consistent with the rest of the Java system***

Because of Java's acceptance in the user community thus far, the designers feel that they should not stray from the current design of the core Java system.

5. ***Keep it simple***

This goal probably appears in all software design goal listings. JDBC is no exception. Sun felt that the design of JDBC should be very simple, allowing for only one method of completing a task per mechanism. Allowing duplicate functionality only serves to confuse the users of the API.

6. ***Use strong, static typing wherever possible***

Strong typing allows for more error checking to be done at compile time; also, less error appear at runtime.

7. ***Keep the common cases simple***

Because more often than not, the usual SQL calls used by the programmer are simple `SELECT`'s, `INSERT`'s, `DELETE`'s and `UPDATE`'s, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible.

Finally we decided to proceed the implementation using Java Networking.

And for dynamically updating the cache table we go for MS Access database.

Java has two things: a programming language and a platform.

Java is a high-level programming language that is all of the following

Simple

Object-oriented

Distributed

Interpreted

Robust

Secure

Architecture-neutral

Portable

High-performance

multithreaded

Dynamic

Java is also unusual in that each Java program is both compiled and interpreted. With a compile you translate a Java program into an intermediate language called Java byte codes the platform-independent code instruction is passed and run on the computer.

Compilation happens just once; interpretation occurs each time the program is executed. The fig. illustrates how this works.

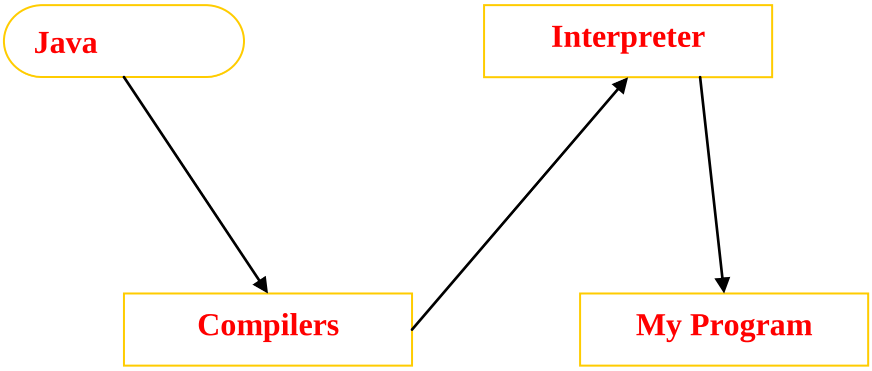


Fig. 5: java program

You can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it’s a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware.

Java byte codes help make “write once, run anywhere” possible. You can compile your Java program into byte codes on my platform that has a Java compiler. The byte codes can then be run any implementation of the Java VM. For example, the same Java program can run Windows NT, Solaris, and Macintosh.

Map Visualizations

Charts showing values that relate to geographical areas. Some examples include: (a) population density in each state of the United States, (b) income per capita for each country in Europe, (c) life expectancy in each country of the world. The tasks in this project include:

Sourcing freely redistributable vector outlines for the countries of the world, states/provinces in particular countries (USA in particular, but also other areas);

Creating an appropriate dataset interface (plus default implementation), a rendered, and integrating this with the existing XYPlot class in JFreeChart;

Testing, documenting, testing some more, documenting some more.

Time Series Chart Interactivity

Implement a new (to JFreeChart) feature for interactive time series charts --- to display a separate control that shows a small version of ALL the time series data, with a sliding "view" rectangle that allows you to select the subset of the time series data to display in the main chart.

Dashboards

There is currently a lot of interest in dashboard displays. Create a flexible dashboard mechanism that supports a subset of JFreeChart chart types (dials, pies, thermometers, bars, and lines/time series) that can be delivered easily via both Java Web Start and an applet.

Property Editors

The property editor mechanism in JFreeChart only handles a small subset of the properties that can be set for charts. Extend (or re implement) this mechanism to provide greater end-user control over the appearance of the charts.

System Configuration

H/W System Configuration:

Processor - Pentium –III

Speed	-	1.1 Ghz
RAM	-	256 MB(min)
Hard Disk	-	20 GB
Floppy Drive	-	1.44 MB
Key Board	-	Standard Windows Keyboard
Mouse	-	Two or Three Button Mouse
Monitor	-	SVGA

S/W System Configuration:

Operating System	:	Windows95/98/2000/XP
Application Server	:	Tomcat5.0/6.X
Front End	:	HTML, Java, Jsp
Scripts	:	JavaScript.
Server side Script	:	Java Server Pages.
Database	:	Mysql
Database Connectivity	:	JDBC.C

CHAPTER 3: ARCHITECTURE

The design of the proposed system discusses the various requirements that will make up the application. Installation on Windows requires WinPcap software which can be downloaded from winPcap website. Jpcap is a set of Java classes which provide an interface and system for network packet capture, it is required for packet capture in Java and built upon Libpcap which is a packet capture library in C language. Java Runtime Environment (JRE) 5.0 or higher will also be required to run this Java application. More space may be required to store the captured packets since the required space on hard disk for installation is less than 1MB.

By conducting the requirements analysis we listed out the requirements that are useful to restate the problem definition

Data Flow Diagram / Use Case Diagram / Flow Diagram

The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system.

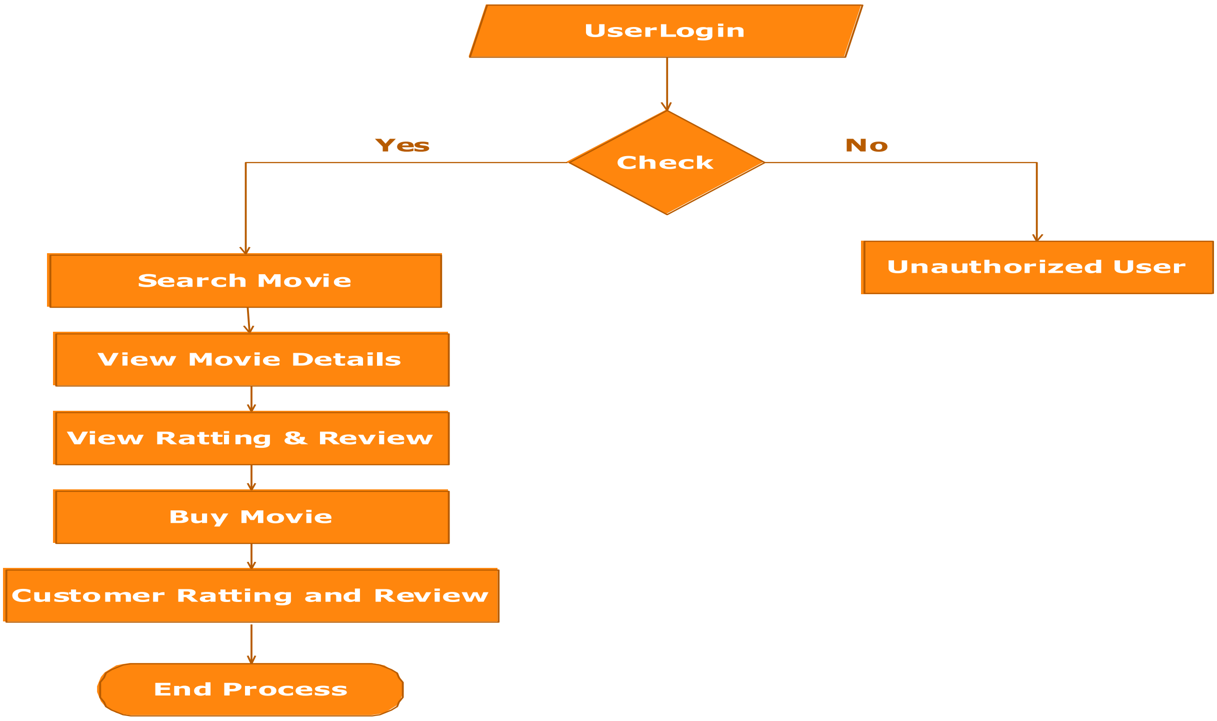


Fig. 6: data flow diagram(user)

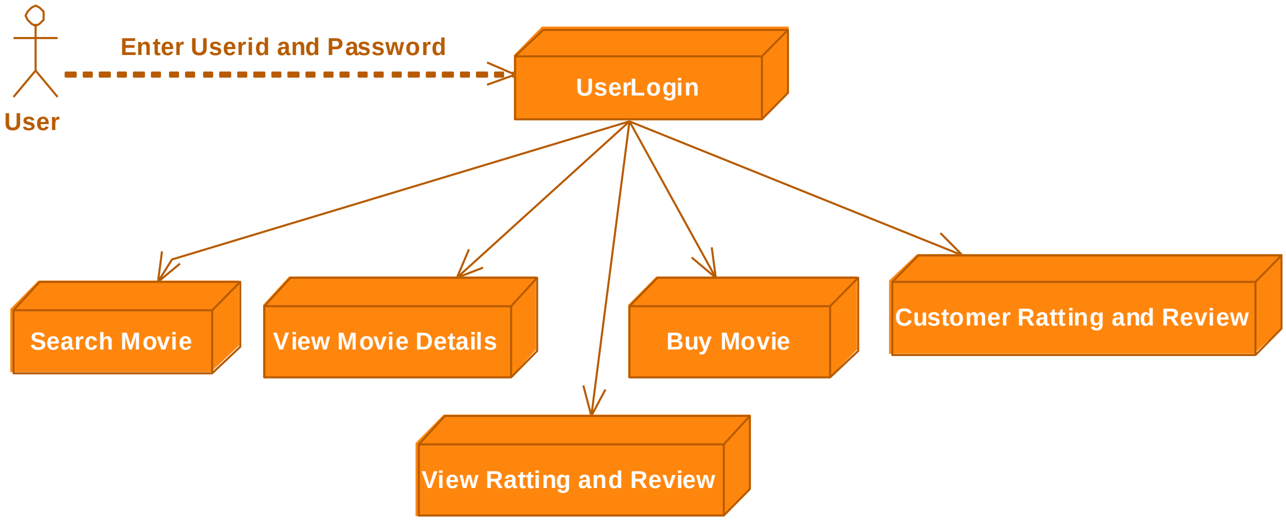


Fig. 7: component diagram

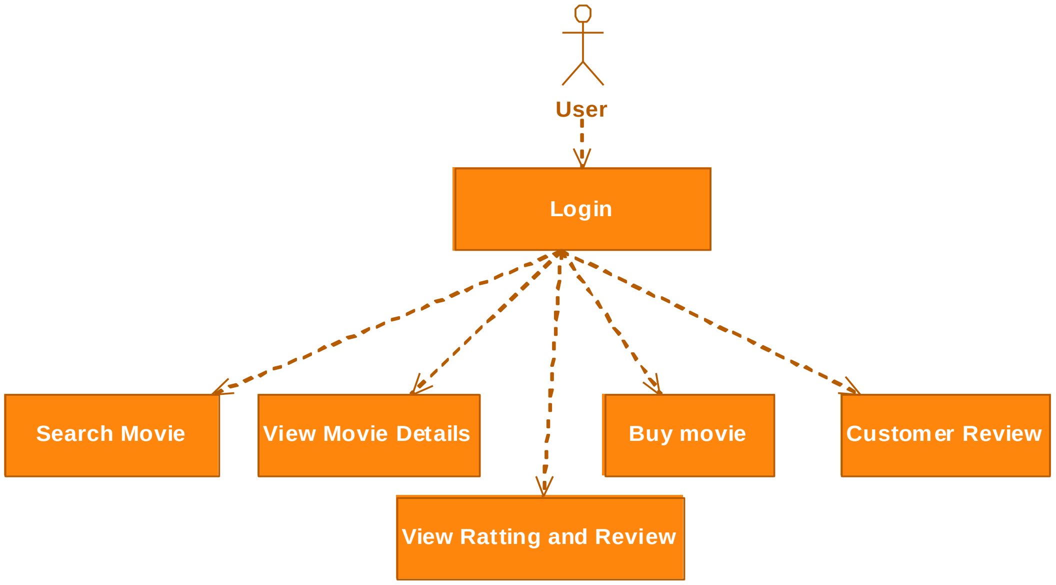


Fig. 8: use case diagram

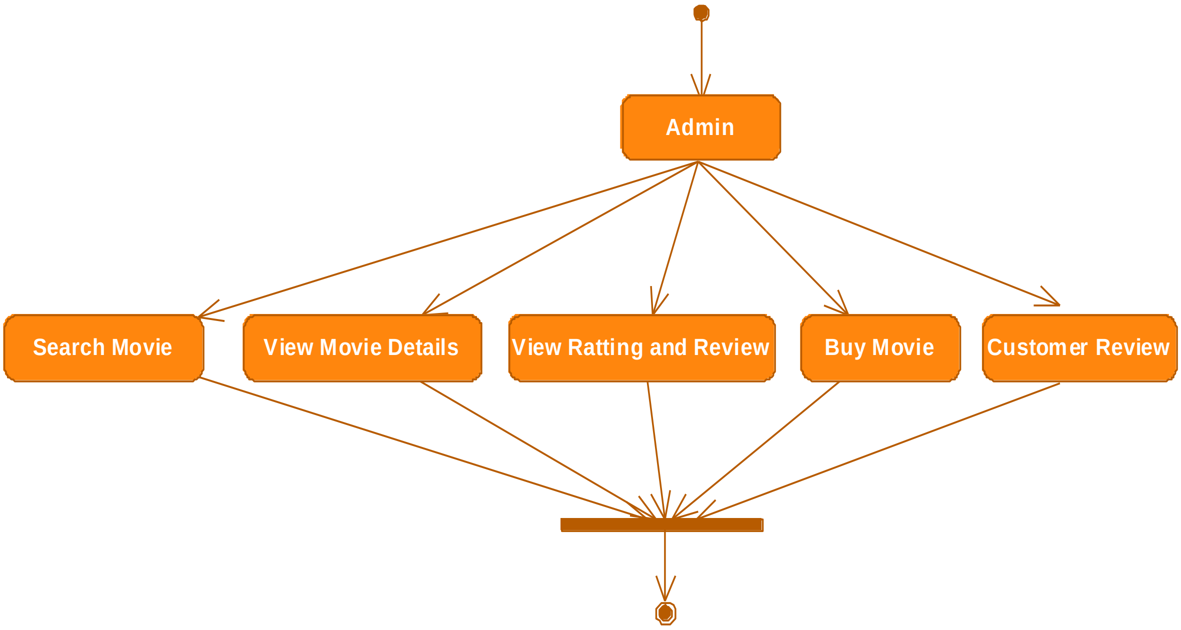


Fig. 9: activity diagram

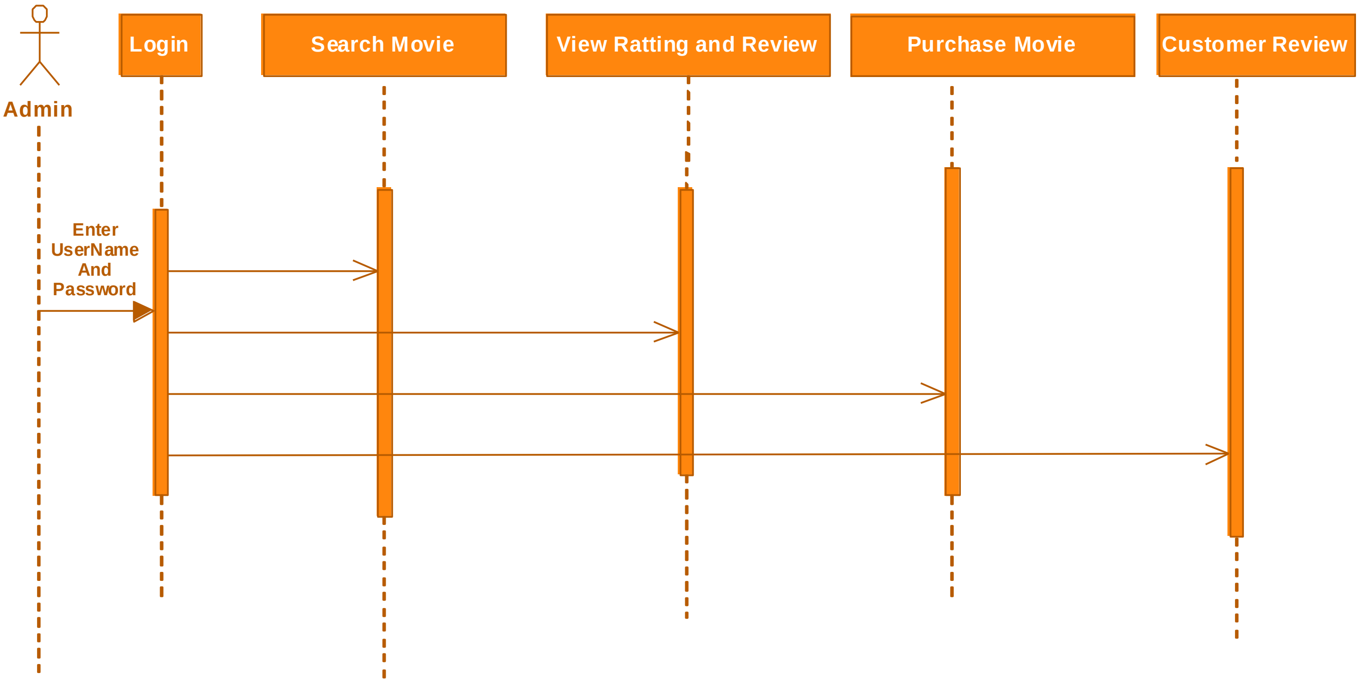


Fig. 10: sequence diagram

CHAPTER 4: SYSTEM DESIGN

4.1: MODULES:

1. Sentiment Analysis
2. Feature-Based Summarization
3. Dataset
4. Opinion-Word Identification

Modules Description

1. Sentiment Analysis

Since a document is composed of sentences and a sentence is composed of terms, it is reasonable to determine the semantic orientation of the text from terms. As a result, the sentiment analysis research started from the determination of the semantic orientation of the terms. Hatzivassiloglou and McKeown employed textual conjunctions such as “fair and legitimate” or “simplistic but well-received” to separate similarly connoted and oppositely connoted words. Esuli and Sebastiani proposed to determine the orientation of subjective terms based on the quantitative analysis of the glosses of such terms, i.e., the textual definitions that are given in online dictionaries. The process is based on the assumption that terms with similar orientation tend to have “similar” glosses (i.e., textual definitions). Thus, synonyms and antonyms could be used to define a relation of orientation. Esuli and Sebastiani described SENTIWORDNET, which is a lexical resource in which each WordNet synset is associated with three numerical scores, i.e., Obj(s), Pos(s), and Neg(s), thus describing how objective, positive, and negative the terms contained in the synset.

2. Feature-Based Summarization

In product-review summarization, people are interested in the reasons why this product is worth buying rather than the principal meaning of the comment. Thus, feature-based summarization is used in movie-review summarization. The feature-based summarization will focus on the product features on which the customers have expressed their opinions. In addition to product features, the summarization should include opinion information about the product; therefore, product features and opinion words are both important in feature-based summarization. As a result, product features and opinion-word

identification are essential in feature-based summarization.

3. Product-Feature Identification

We propose an LSA-based product-feature-identification algorithm and system can obtain a semantically related feature set for each seed. We compared three product-feature-identification approaches, i.e., ratting about product feature, price and delivery.

4. Opinion-Word Identification

In addition to feature identification, opinion words about the product features are important as well. Hu and Liu extracted the opinion words by retrieving the nearby adjective of product features. In addition to language sentence-structure characteristic, Zhuang *et al.* used the dependency grammar graph to find out some relations between feature words and the corresponding opinion words in training data. They both rely on language sentence structure to extract opinion words; therefore, these approaches will be applicable to those language sentences having such a characteristic.

4.2: SYSTEM STUDY

4.2.1: FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ♦ ECONOMICAL FEASIBILITY
- ♦ TECHNICAL FEASIBILITY
- ♦ SOCIAL FEASIBILITY

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

CHAPTER 5: METHODOLOGY

Methodologies used :

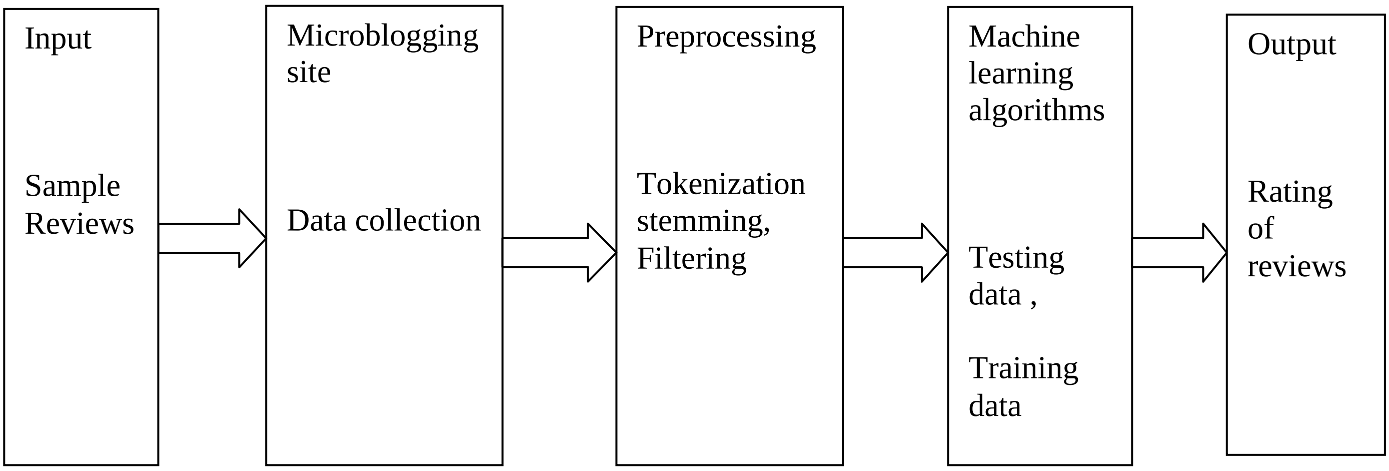


Fig. 11: flowchart of the program

5.1: Data collection:

The first work in this thesis is to collect huge amount of data collection of data although is not a simple task as it may seem to anyone at first thought. For this we have to make certain assumption and decision. Basically we have to collect two different data sets: test data, training data.

Test data Collection: One of the objective of this thesis is to analyze the sentiment of messages posted in reaction to movie reviews, so we have to collect tweets about movies only. However , it is not a simple task.

A corpus of around 1000 tweets about movie reviews was sampled from the test data and we manually examined and annotated that data as positive or negative because the data which we collected was unlabelled ,so we formed two classes as pos, neg.

Out of thousand post that have been annotated 31 posts here needed context to understand and determine their sentiments. thus 97% of our test data did not require context to determine their sentiments. Many a times in a time came when it was difficult to determine the sentiment, as they seemed both positive and negative.

4. **Training data:**

We have chosen only subjective data for training ,no objective was collected here.

Subjective data involves positive and /or negative sentiment only. if does not consists of any neutral tweets.

A total of 5000 subjective twitter posts were collected for corpus. One this data was collected, it was separated into tweets that contain only negative emoticons tweets that contain only positive emoticons and tweets that contain both emoticons.

Class	Training data	Test data
Negative	2000	400
Positive	3000	600

Table 1: data collection

5.3:Supervised approaches

This is where most of time and effort was spent in our thesis under this section, different supervised machine learning approaches were used and explained. To be able to experiment with different machine learning algorithms and to able the identification of factors that affect our results ,we used a three step process.

The first step is the preprocessing of the training data. The second is feature extraction and data representation in a particular format.(semantic vsm here).

The third is the classifier training and testing phase with different machine learning algorithms .

this approaches helped in experimenting with different possibilities by varying the preprocessing operations, feature extraction and then using the machine learning algorithms .each of this will be explained in the following subsections.

5.3.1:Pre-processing:

Cleaning of the data:

Since the corpus we are taking consists of syntactical features that may not be useful for machine learning the data needs to be cleaned for further use

Tokenization: in this process , we have broken the stream of text up into words , phrases ,symbols and then called them as token .the frequency of a particular token in the whole data set is also counted. With that we have also taken out total no. of unique words in our data set .

Stemming: in this process we have reduced the inflicted words to their stem or root word. For this we have taken in to account that stem need not to be identically same to the morphological root of the word ; it is sufficient that a related words could map to the same stem is not itself a valid root

Stop word removal : after tokenization we are having a big list of words occurring in data sets and mostly not all of them are useful for learning task. It is imperative to reduce the size of feature space as far as possible. So stop word removal step will be done prior too tokenization to remove all occurrence of these useless words like ‘a’,’ an’, ’the’, ’is’, etc .

Filtering : during filtering process various sub tasks were done .

url :url were removed ,in order to remove the feature size during training

emoticons : all emoticons were replaced by pos and neg classes

username and hash tags : we replaced them with <un> and <ht> respectively.

Removal of repeated comments

Lower casing : all characters were lower cased to ensure that all tokens map to the corresponding feature irrespective of casing .

Class	Data	After duplicate removal	Duplicates
Positive	3000	2960	40
Negative	2000	1990	10
Total	5000	4950	50

5.3.2: Removing duplicates: duplicates were removed from all training data as they just add Weight and nothing else .only exact duplicates were removed by us .the result of the removal of duplicates are given in table as can be seen from the table there were duplicates present in each class . the no. of duplicates is more in positive tweets then there are in negative tweets ,because people have tendency to re tweet something they are very happy with the total percentage of duplicates is 10% only.

5.3.3:Feature extraction and instance representation : the most important question in machine learning is how to represent data both the training and test data must be represented in some or the other way in order for a machine learning algorithm to learn and thus build a model out of it .some of the ways in which the data can be represented could be feature based or bag of words representation .by features based ,it is meant that some attributes which we think that they capture the pattern of the data are selected first and the entire data set must be represented n terms of them before it is fed to any machine learning algorithm.

Different features like n-gram presence or n-gram frequency ,pos tags ,syntactic features or semantic features can be used here in this representation .

This paper explores and designs a system for movie rating and review summarization in which semantic orientation of comments. System response time are considered. Practically, when we are not familiar with a specific product, we ask our trusted sources to recommend one. Today, the popularity of the Internet drives people to search for other people’s opinions from the Internet before purchasing a product or seeing a movie. Many websites provide user rating and commenting services, and these reviews could reflect users’ opinions about a product. For example, the customer-review section in Amazon.com lists the number of reviews, the percentage for different ratings, and comments from reviewers.

When people want to purchase books, CDs, or DVDs, these comments and ratings usually influence their purchasing behaviors. In addition to these websites, a search engine is another important source for people to search for other people’s opinions. When a user enters a query into a search engine, the search engine examines its index and provides a listing of best-matching web pages according to its criteria, usually with a short summary containing the document’s title and, sometimes, parts of the text. In this paper, we collected movie reviews from Internet Blogs that do not consist of any rating information. Sentiment analysis is performed to determine the semantic orientation of the reviews and movie-rating score is based on the sentiment-analysis result.

5.3.4: Stemming:

A stemming algorithm is a computational procedure which reduces all words with the same root (or, if pre-fixes are left untouched, the same stem) to a common form, usually by stripping each word of its derivational and inflectional suffixes. Researchers in many areas of computational linguistics and information retrieval find this a desirable step, but for varying reasons. In automated morphological analysis, the root of a word may be of less immediate interest than its suffixes, which can be used as clues to grammatical structure. (See, e.g., Earland Resnikoff and Dolby .This field has also been reported on by S. Silver and M. Lott, Machine Translation Project, University of California, Berkeley [personal communication].) At the other extreme, what suffixes are found may be subsidiary to the problem of removing them consistently enough to obtain sets of exactly matching stems. Word-frequency counts using stems, for stylistic (as described by S. Y. Sedelow [per- sonal communication]) or mathematical analysis of a body of language, often require matched stems. (So does stemming as part of an information-retrieval sys- tem, the specific application which motivated this pa- per.) But certain linguistic problems are common to any "stem-oriented" stemming algorithm, no matter what its ultimate use. The brief description below of the framework within which Project Interrex is planning to use a stemming algorithm should be viewed as but one possible application for research on the morphological structure of English and other languages. Similarly, a variety of applications are considered in evaluating the theoretical and practical attributes of several previous algorithms. As a major part of its information transfer experiments, Project Intrex is developing an integrated re- trieval system in which a library user, through a remote computer terminal, can first obtain extensive information from a central digital store about documents that are available on a specific subject, and then obtain the full text of the documents. A prototype retrieval system is being assembled in order to permit experimentation with its various components. The experimental system will use a specially compiled augmented library catalogue containing information on approximately 10,000 documents in the field of materials science and engineering, including not only author, title, and other basic data about each document but also an abstract, bibliography, and a list of subject terms indicating the content of the document. Each subject term is a phrase of one or more English words. A stemming algorithm will be used to maximize the usefulness of the subject terms.

In many cases, the information which is semantically significant to the user of the system is contained in the stems of the lexical words in the subject terms, and suffixes and function words merely enable this information to be expressed in a grammatical form. The form of the words which the user inputs will often not correspond to that of the original words in the catalogue. To Permit the words in the user's query to match the words in the catalogue entry's subject terms, both query and subject terms can be stripped of the suffixes that prevent their matching. For example, computational and computing might both be stemmed to compute.

In constructing the software needed for this particular application of stemming (or any other), we encounter questions which are answerable only in terms of the over-all system. For instance, what should constitute a "word" to be stemmed? In the case of Intrex, what suffixes should the algorithm search for that are specifically 22 oriented toward terms in materials science and engineering? These are questions of less general interest than the linguistic problems of extracting a stem from any one word in a non-specialized vocabulary (for an example of lists of affixes taken from terms in specific technical fields, see Dyson). The development of an efficient algorithm should logically precede investigation of these questions, and they will not be discussed further here. The approach to stemming taken here involves a two- phase stemming system. The first phase, the stemming algorithm proper, retrieves the stem of a word by removing its longest possible ending which matches one on a list stored in the computer. The second phase handles "spelling exceptions," mostly instances in which the "same" stem varies slightly in spelling according to what suffixes originally followed it. For example, absorption will be output from phase one as absorpt, absorbing as absorb. The problem of the spelling exceptions, which in the above example involves matching absorpt and absorb, is discussed thoroughly in Section V of this paper. One particular solution to the problem, termed recoding, has been implemented in the present phase two. We also plan to use the present basic algorithm as a foundation in testing out other feasible solutions.1 This plan is appropriate because spelling-ex- ception rules can, and probably should, be formulated independently of the stemming algorithm proper.

II. Stemming, Form, and Meaning By its computational nature, a stemming algorithm has inherent limitations. The routine handles individual words: it has no access to information about their gram- matical and semantic relations with one another. In fact, it is based on the assumption of close agreement of meaning between words with the same root. This assumption, while workable in most cases, in English represents an approximation at best. It is a better or worse approximation depending on the intended use of the stems, the semantic vagaries of individual roots, and the strength of the algorithm (how radically it transforms words). A stemming algorithm strong enough to group together all words with the same root may be unsuitable for, say, word-frequency counting. For such applications one would not wish a pair like neutron-.neutral- izer to coincide, and one would prefer to work with a very limited list of suffixes. Where stems are used as a means of associating related items of information, as they are in an automated library catalogue, and where the catalogue can be interrogated in an on-line mode, it seems best to use a strong algorithm, that is, one that will combine more words into the same group rather than fewer, thus pro- viding more document references rather than fewer. I am indebted to Richard S. Marcus and Peter Kugel for valuable discussion of this specific problem and of this report as a whole. After a word in the library user's query has been stemmed and a matching stem and associated list oriented toward terms in materials science and engineering? These are questions of less general interest than the linguistic problems of extracting a stem from any one word in a non-specialized vocabulary (for an example of lists of affixes taken from terms in specific technical fields, see Dyson). The development of an efficient algorithm should logically precede investigation of these questions, and they will not be discussed further here.

The approach to stemming taken here involves a two- phase stemming system. The first phase, the stemming algorithm proper, retrieves the stem of a word by removing its longest possible ending which matches one on a list stored in the computer. The second phase handles "spelling exceptions," mostly instances in which the "same" stem varies slightly in spelling according to what suffixes originally followed it. For example, absorption will be output from phase one as absorpt, absorbing as absorb. The problem of the spelling exceptions, which in the above example involves matching absorpt and absorb, is discussed thoroughly in Section Vof this paper. One particular solution to the problem, termed recoding, has been implemented in the present phase two. We also plan to use the present basic algorithm as a foundation in testing out other feasible solutions.1

This plan is appropriate because spelling-exception rules can, and probably should, be formulated independently of the stemming algorithm proper

II. Stemming, Form, and Meaning By its computational nature, a stemming algorithm has

inherent limitations. The routine handles individual words: it has no access to information about their grammatical and semantic relations with one another. In fact, it is based on the assumption of close agreement of meaning between words with the same root. This assumption, while workable in most cases, in English rep- resents an approximation at best. It is a better or worse approximation depending on the intended use of the stems, the semantic vagaries of individual roots, and the strength of the algorithm (how radically it transforms words). A stemming algorithm strong enough to group together all words with the same root may be unsuitable for, say, word-frequency counting. For such applications one would not wish a pair like neutron-.neutralizer to coincide, and one would prefer to work with a very limited list of suffixes.

Where stems are used as a means of associating re- lated items of information, as they are in an automated library catalogue, and where the catalogue can be in- terrogated in an on-line mode, it seems best to use a strong algorithm, that is, one that will combine more words into the same group rather than fewer, thus providing more document references rather than fewer. After a word in the library user's query has been stemmed and a matching stem and associated list of ull-word forms has been found in the catalogue and presented to the user, he may decide to discard some of these forms in order to inhibit searching for those full- word forms which are unrelated to his subject.

Occasionally, the output of a stemming routine may be not only ambiguous but also "not English." This happens when a suffix is identical to the end of some root. For instance, -ate is a noun suffix in directorate, but simply part of a verbal root in create and appreciate. In English, situations of this type limit the use of suf- fixes as clues to parts of speech. Sometimes grammatical information is required for stemming, not provided by it. However, the generation of such non linguistic stems as cre - and appreci- is not a serious problem; if the pur- pose of stemming is only to allow related words to match, then the stems yielded by a stemming algorithm need not coincide with those found by a linguist. The exact form of the stem is not critical if it is the same no matter what suffixes have been removed following it, and if "mistaken" stemming does not generate an ambiguity. Similarly, the ending that must be removed in order to achieve a consistent algorithm is determined in relation to the stemming system as a whole. The ending may or may not be exactly equivalent to some entity in English morphology, and it may be acceptable to have the computer program remove it when a linguist would not, with no detriment to the ultimate results.

III. Types of Stemming Algorithms

Two main principles are used in the construction of a stemming algorithm: iteration and longest-match. An algorithm based solely on one of these methods often has drawbacks which can be offset by employing some combination of the two principles.

Iteration is usually based on the fact that suffixes are attached to stems in a "certain order, that is, there exist order-classes of suffixes (see, e.g., Lejnieks [4]). Each order-class may or may not be represented in any given word. The last order-class—the class that occurs at the very end of a word—contains inflectional suffixes such as -s, -es, and-ed. Previous order-classes are derivational.(As pointed out by J. L. Dolby [personal communication], there are several cases known in which a derivational suffix (-ness) follows an inflectional one (-ed or -ing). This occurs with certain nominalized adjectives derived from verbs by use of one of these two inflectional endings, for example, relatedness, disinterestedness, willingness.) An example of the lowest order-class in a word may be what is technically part of the root (see the -ate example above), but for the purposes of computation it is considered part of the ending. An iterative stemming algorithm is simply a recursive procedure, as its name implies, which removes strings in each order-class one at a time, starting at the end of a word and working toward its beginning. No more than one match is allowed within a single order-class, by definition. One must decide how many order-classes there should be, which endings should occur in each, and whether or not the members of each class should be internally ordered for scanning.

The longest-match principle states that within any given class of endings, if more than one ending provides a match, the one which is longest should be removed. This principle is implemented by scanning the endings in any class in order of decreasing length. For example, if -ion is removed when there is also a match on -ation, provision would have to be made to remove -at, that is, for another order-class. To avoid this extra order-class, -ation should precede -ion on the list. An algorithm based strictly on the longest-match principle uses only one order-class. All possible combinations of affixes are compiled and then ordered on length. If a match is not found on longer endings, shorter ones are scanned. The obvious disadvantage to this method is that it requires generating all possible combinations of affixes. A second disadvantage is the amount of stor- age space the endings require.

The first disadvantage may also be present to a large degree when one is setting up an iterative algorithm with as many order-classes as possible. To set up the order-classes, one must examine a great many endings. Furthermore, it is not always obvious to which class a given string should belong for maximum efficiency. It is also entirely possible that the occurrence of members of some classes is context dependent (see below). In short, while an iterative algorithm requires a shorter list of endings, it introduces a number of complications into the preparation of the list and programming of the routine.

Some idea of the breadth of these complications is gained through consideration of another basic attribute of a stemming algorithm: it is context free or context sensitive. Since "context" is used here to mean any attribute of the remaining stem, "context free" implies

no qualitative or quantitative restrictions on the removal of endings. In a context-free algorithm, the first ending in any class which achieves a match is accepted. But

here should presumably be at least some quantitative restriction, in the sense that the remaining stem must not be of length zero. An example of this extreme case is the matching of -ability to ability as well as to com- putability. In fact, any useful stem usually consists of at least two letters, and often three or four constitute a necessary minimum. The restriction on stem length varies with the ending; how it varies can again only be determined in relation to the total system. The algorithm developed by Professor John W. Tukey of Prince- ton University (personal communication) associates a lower limit with each ending. Some of his limits are quite high (e.g., seven letters). I have been less conservative and have proposed a minimum stem length of two; certain endings have an additional restriction in that their minimum stem length is three, four, or five letters.

The kind of qualitative contextual restrictions that should be imposed is a somewhat open question. In order to get the best results, certain endings should not be removed in the presence of certain letters in the resultant stem, usually those letters that immediately pre- cede the ending. The more desirable form of context- sensitive rule is a general one that can be applied to a number of endings, but such rules are few. One example is "do not remove an ending that begins with -en- ,following -e." Violation of this rule would change seen to se-, a potentially ambiguous stem (cf. sea minus -a, seize minus- ize, etc.). But a number of rules must be created for individual endings in order to avoid certain special cases peculiar to those endings. One can go to great lengths in this direction, with increasingly small returns. I have preferred to start by treating a number of the more obvious exceptions in the hope that the percentage of words not accounted for will be small enough to preclude the need to add many additional rules. An iterative stemming algorithm, that is, one that contains more than one order-class of endings, is presumably no less complicated by context-sensitive rules than a one-class algorithm, and is probably more so; exceptions associated with the members of each class may depend on a rather complicated context. For ex- ample, suppose there is a rule (in a non-iterative algorithm) stating that minimal stem length is five before -ionate. The endings -ion and -ate occur separately, also, with different restrictions. In an iterative routine, -ion and -ate would only occur as separate endings, in different order-classes; and -ion would be restricted by the rule that its preceding context must be of length five if -ate was found during the preceding iteration. In other words, the endings that are removed may influence the lower-order endings that can be removed subsequently. The implications for simplicity in programming are self-evident. In a pure longest-match algorithm, the only context that need be considered is the prospective stem itself.

Since computer-storage space for endings was not an immediate problem, it was decided to test a non-iterative stemming algorithm based on a one-class list of endings. That is, the intuitively inefficient procedure of listing both singular and plural forms, and so on, has been followed in order to minimize the number of con- text-sensitive rules necessary. Compilation of the actual
list of endings used is discussed in the next section; the algorithm is outlined in Section VI.

The author is aware of three previous major attempts to construct stemming algorithms. Tukey has proposed a context-sensitive, partially iterative stemming algorithm whose endings are divided into four order-classes. The first (highest-order) class contains only terminal s which, however, is not removed after i, s, or u. The second class is recursive, the third is non-recursive and ordered on length. The fourth class consists of remaining terminal consonants. The last three classes also have a few members each with simple context restrictions, and all classes have limits on minimum stem length. (The basic structure of this "tail-cropping" algorithm is not affected by its multilingual orientation, though the endings used would obviously differ from those found in a procedure for English only.) One of the more interesting things about the Tukey system is its structural complexity. One class uses the longest-match principle only, while another is iterative (and thus not a proper order-class). Presumably the object of this heterogeneous structure is to avoid the repetitiveness of a one-class ending list in the most concise way possible. However, as stated earlier, there is a compromise between conciseness of rules and simplicity of programming. By contrast, the algorithm developed at Harvard University by Michael Lesk, under the direction of Professor Gerard Salton [10], is based on an iterated search for a longest-match ending. After no more matches can be found, terminal i, a, and e are removed, and then possibly terminal consonants. There are apparently no con- textual restrictions of any kind. (A brief description of the algorithm, including a useful list of 194 endings, was transmitted to us via personal communication. A sample of these suffixes, and further information about the algorithm, have more recently appeared in Salton.)

A third algorithm has been developed by James L. Dolby of R and D Consultants, Los Altos, California (personal communication). This algorithm works in three stages, the first of which involves a set of context- dependent transformations. Most of the cropping is done in the second stage, a context-free, longest-match, re- cursive procedure which removes endings in any order but is subject to the restriction of a two-syllable mini- mum stem length. In the final stage there is a context dependent dropping of inflectional forms. The endings used were derived by algorithm from word lists on the basis of orthographic context, and are "minimal" segments of one to four letters in length.

IV. Compilation of a List of Endings

A one-class list of endings (concatenations of suffixes) was compiled in the following way: A preliminary list was based on endings found in a small portion of the augmented catalogue being developed by Project Interrex and on endings in the list used at Harvard. The preliminary list was evaluated by applying the endings on this list to a portion of the output from Tukey's tail- cropping routine, levels 1-3, and volumes 5-7 of the Normal and Reverse English Word List [8] (volumes 5 7 contain unbroken words sorted alphabetically when written from right to left). Since each of these lists is organized according to ends of words, it was possible to see whether the removal of a given ending would result in (1) two different stems matching, or (2) a stem not matching another stem which it should match. Either of these conditions, unless it was caused by a spelling exception or caused improper matching in only a few rare cases, necessitated the addition of new endings, the disposing of old ones, or the addition of context- may occur, but do not always occur. The second assumption is that these changes involve no more than two letters at the end of a stem--this is merely an empirical result which has not yet been contradicted. It has also been observed that the sequences of letters that cause difficulty are often common to more than one class of exceptions. In recoding, this means that some rules can cover more than one type of exception, although it is not usually the case.

The crucial difference between recoding and partial matching is this: a recoding procedure is part of the stemming algorithm while a partial-matching procedure is not. Partial matching operates on the output from the stemming routine at the point where the stems derived from catalogue terms are being searched for matches to the user's stemmed query. All partial matches, within certain limits, are retrieved rather than just all perfect matches; discrepancies are resolved after retrieval, not in the previous stemming procedure. This has the advantages of reducing stemming to the one-step process of removing an ending and of eliminating the context specifications sometimes

needed in recoding. The dis- advantages, which are not so obvious, can be discussed only after a more complete description of a partial matching procedure is given. Such a procedure starts with an unmodified stem S1
—again, absorption is a good example. The first step is to search the list of stemmed catalogue terms for all those which begin with S1 minus its last two letters: in this
case, all stems of any length beginning with absor, which we call S2. Of course, special provisions will have to be made for cases in which S1 is only two or three letters long. Among those stems returned will be absorpt and absorb. Absorbefaci, the stem of absorbefacient, may also be found. This last item will be eliminated, probably for the better, by the next step of the procedure, which discards all stems more than two characters longer than S1 (here, more than nine letters long). We then have collected all stems which match absorpt within two letters in either direction. Given any one of these, Sj, a final match is allowed between Sj and S1 if and only if either Sj = S1 or the following conditions are satisfied:

1. The stems Sj and S1 must match at least up to two letters before the end of the longer of them.
2. If Sj and S1 are the same length and differ by one letter, this letter plus a blank must occur on a closed list (see Appendix D) for each stem.
3. If Sj and S1 are the same length and differ by two letters, each sequence of two letters must occur on the list.
4. If Sj and S1 differ in length by one, the last two letters of the longer, and the last of the shorter plus a blank, must occur on the list.
5. If Sj and S1 differ in length by two, the last two letters of the longer must occur on the list. The above rules amount essentially to examining the last two letters of stems that match up to that point; if the stems are different lengths, all "missing letters" in the shorter are represented by blanks. The "closed list" needed for this routine is given in Appendix D.

It may appear that an unacceptable number of "wrong" matches would result from this procedure, since there are no restrictions on which pairs of items on the list may be used to produce a match. There are two defenses against this view:

First, such a closed list does exist. Many partial matches will *not* be allowed. Of those that are allowed erroneously, many would have been produced also by a recoding procedure, for much the same reasons.

Second, we can make a probabilistic argument. Most of the stems used will probably be fairly

long—long enough so that there are unlikely to be many matches within two letters. Any S_j found by searching with S_2 stands a good chance of being related to S_2, and thus
to S_1.

In short, while a partial-matching procedure may produce *no fewer wrong* matches than recoding, it will probably produce *more right* ones. It is inherently more flexible than recoding rules; all classes of exceptions do not have to be specified beforehand. Part of this flexi-bility results from allowing S_1 and S_j to differ in length by two letters in *either* direction. Yet this condition also provides a built-in barrier against certain types of wrong matches, as the following example illustrates:

Convex is recoded to convic by the rule *ex* → *ic; convict,* the stem of *conviction,* is recoded to *convict by* the rule *ct* → *c.* This erroneous match is *not* allowed in partial matching, since although condition (4) is satisfied, condition (1) is not.

Partial matching is a kind of controlled recoding; the recoding takes place *only* if a partial, but not complete, match is found. The original stem is still preserved, however, providing a constant check for violation of condition .

Using partial matching as a substitute for recoding does have one major disadvantage for a system using disk storage, as Interrex does, and it is a potentially serious one. In some cases, the time-consuming retrieval from the disk of a great number of partial matches, those beginning with S_2, will be necessary. These cases are most likely to occur with very short stems. The question is whether in such instances S_2 can be lengthened (made closer to S_1) enough to avoid this problem and still retrieve all acceptable matches. Empirical data are needed to answer this question, as well as to determine whether the number of short stems used is great enough to warrant concern. Any timing, programming, or other complications which partial matching introduces must be small enough to be balanced out by other advantages it may offer.

VI. The Two-Phase Stemming Routine and Its Results

Several progressively more advanced versions of the Interrex stemming routine have been coded in AED (a compiler language developed at the Electronic Systems Laboratory) [7, pp. *367-85]* and run on sample batches of words, using the MIT 7094 CTSS system. The flow chart in Fig. 1 shows the most important features of the stemming and recoding parts of the program.

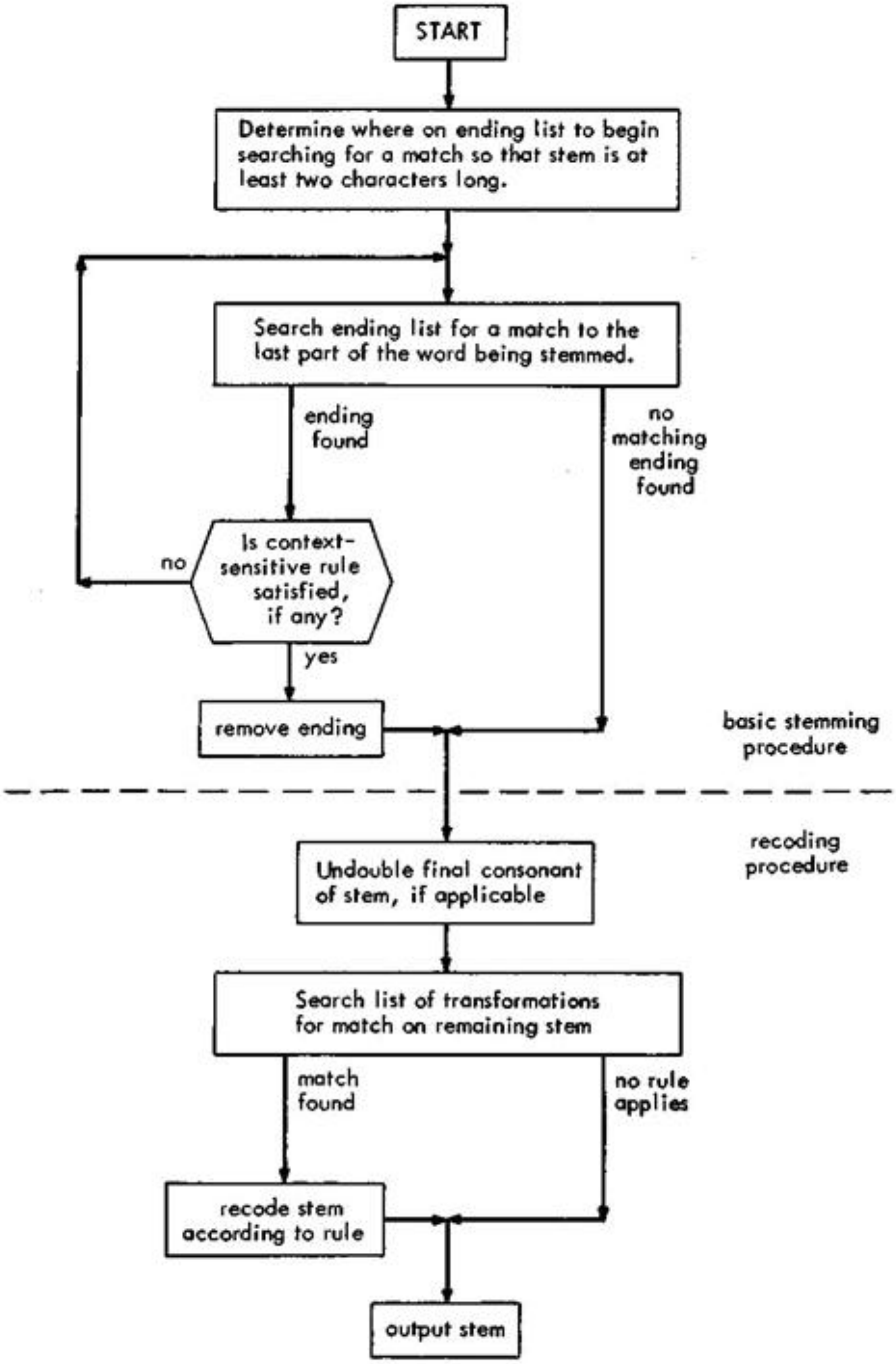

START
Determine where on ending list to begin searching for a match so that stem is at least two characters long.
Search ending list for a match to the last part of the word being stemmed.
ending found
no matching ending found
Is context-sensitive rule satisfied, if any?
no
yes
remove ending
basic stemming procedure
recoding procedure
Undouble final consonant of stem, if applicable
Search list of transformations for match on remaining stem
match found
no rule applies
recode stem according to rule
output stem

Fig. 12: n-gram stemming

While a full evaluation of this stemming system within the Project Intrex environment will not be possible until the augmented catalogue data base is completed, output so far indicates that the procedures used are workable and will yield very good results with only minor changes. These changes involve the list of endings and occasionally the recoding rules; the types of operations performed remain the same.

To give some idea of the alterations that are needed to make the system highly effective, I shall discuss several of the changes that have been made in the pro-gram. Fig. 2 shows the result of stemming several groups of related words. An obvious problem was that "magnet" and "magnesium" had the same recoded stem. This problem was easy to fix by changing recoding rule 32 from *et* → *es* to *et* → *es* except following *n.*

An additional recoding rule took care of the discrepancy between *meter*→ *meter* and *metric* → *metr:metr*
→ *meter.* All other changes involved the stemming procedure: -*ium*, -*ite,* and -*itic* were added to the list of endings, with the stipulation that -*ite* be removed only in certain rather limited cases and -*itic* only after *t* or *ll;* the rule governing -*al*- endings was changed so that they are not removed after *met-; l* was added to the list of stem-final consonants to be undoubted; and the context in which the removal of -*on* is allowable was broadened to include single *t.* The results after these changes are shown in Fig. 3. It is expected that several more such evaluations of a random group-sample will catch most similar difficulties still left in the program, although it is likely that minor revisions will be required as long as the vocabulary of the data base continues to increase.

n-gram’s stemming :

Stemming, a very useful tool under the domain of Information Retrieval (IR), supported by almost each and every modern indexing and search systems. Sometimes put under a single roof with its sibling process, lemmatization, stemming is nothing but to reduce the morphological variants of a word into a single form- the stem. Before indexing, truncating attached affixes (suffixes or prefixes) from terms, makes thus generated stem to represent a broader concept than the original term, the stemming process eventually increases the number of retrieved documents in an IR system. Although, the effectiveness of stemmers has always been argued [7, 8], still stemming is kept as an active method in the IR toolbox.

We have a rich literature of stemming process- there are linguistic and statistical approaches. The former are *rule based* techniques [4, 8, 11, 15, 17], in which a priori knowledge of the target language morphology is required. The latter are subject to exhaustive statistical analysis of the corpus, but work in a language-neutral way [5, 6, 12-14, 19]. There are also some lexicon based approaches . Linguistic stemmers have one major disadvantage that they are not suitable multilingual environment. Statistical algorithms, such as one which is based on character frequencies can cope up with this limitation.

In the urn of the statistical techniques, the most interesting and robust technique that caught our attention is *Single N-gram Stemming* by James Mayfield and Paul McNamee [13]. The authors hypothesized that exploiting overlapping sequences of *N* characters, the benefits of stemming can be achieved in a language independent way. They argued that some of the N-grams (actually, low frequent) derived from a word will be common to only those portions of the word that do not exhibit morphological variation, i.e. they can serve as an adequate stem substitute. They suggested selecting the word internal N-gram with the highest inverse document frequency (IDF) as a word's *pseudo stem.* They gave an excellent reasoning that a particular morphological variation being exhibited by affixes tends to be repeated across many different words, and therefore exhibit low IDF. As an instance, they tabled document frequencies of different N-grams of the word 'juggling' (from the CLEF 2002 collection) and the suggested pseudo-4 and pseudo-5 stems, according to their concept.

The key behind our present research resides in the tables being talked above. The pseudo-4 stem for the words currency and warrens were *rren* and *rens* respectively. The pseudo-5 stem for juggler, currency and warrens were *ggler, rency and rrens* respectively. A general insight says that a stem has to start from the beginning character of the word, not from anywhere else. So, is there any scope of improvement over this already elegant method?

Endeavoring to answer the above question is the essence of our present paper. We work for the better utilization of N-grams, keeping the idea of counting their frequencies intact. Rather than depending on the simple concept of IDF, we develop a novel algorithm that gives better approximations of stems.

2. Proposed Method:

The authors reported in their experimental results that out of the raw words, Snowball stems , pseudo stems a grams, the trend was for 4-grams to perform best. They concluded the fact

that 4-grams, the initial 4 characters of any word are the best representation of a word stem. This strengthens our insight that for a valid (not from a strict linguistic point of view) and nicely approximate stem, it should start from the beginning. So what if instead of finding frequencies of overlapping N characters (i.e. words internal N-grams), we approximate stem from the frequencies of 4-gram, 5-gram, 6-gram and so on? Seems a promising direction.

Our idea is to take 4-gram as our initial guess for finding the stem. Obviously, this rules out words with length 3 or less. This would be a stunning advantage as many of the stop words are of length 3 or less. Now, for a given word, what we are left with finding frequencies of 4-gram, 5-gram and so on up to the length of the word and selecting the best from them. Before we embark upon our approach, let us look at some of the different N-gram frequencies from COCA .We start with the same word *juggling*.

Table 2: Different N-gram frequencies from COCA for the word ‘juggling’

N-gram	***Frequency***	***N-gram***	***Frequency***
jugg*	915	jugglin*	328
juggl*	729	juggling*	328
juggli*	328		

We infer that an N-gram of a word that serves as an adequate approximation of the stem must has a relative good frequency across the corpus. All the N-grams less than this must be highly frequent and above this tend to exhibit lower frequencies gradually, ending in the total frequency of the given word (there may be the case where two or more consecutive entries are identical.)Well, above table successfully meets our assumption.

Thus, we have an integer variable in hand whose values tend to decrease (or remain constant) with the increase in the length of a partial word (N-gram) by one unit. It is very likely that this fall is very steep at the transition from invariant part of a word (stem) into the derivational or suffix part. But how can we catch a relative maximum fall? One way is to take deviation of each two consecutive entries. We define a metric as under

$$\lambda_i = \begin{cases} abs(F_i - F_{i\ -1})\ if\ i > 4 \\ 0 \qquad\qquad\qquad else \end{cases}$$

The NEWSPAPER search option was selected and the minimum frequency was set to 10.

Where F_i is the frequency of the ith N-gram. Next we calculate the second order deviation

$$\Delta_i = \lambda_i - \lambda_{i_{-1}} \tag{2.2}$$

Let $\psi_N$be the variable corresponding to the N-gram of length N. We define our procedure in two phases:

Phase 1:

1) Initialization:

$$\lambda_4 = M$$
$$\psi_N = 4$$
where M is a very high integer value

2) Recursion:

Calculate   , $4 < i \leq |word|$

If

λ_i

$\lambda_i > \Gamma$

$$\psi_N = \underset{4<i\leq|word|}{\operatorname{argmax}} [F_i, F_{i-1}]$$

else

$$\psi_N = i$$

3) Termination:

a.) If N=|word|, stop. Else, calculate Δ_i. b.) If Δ_i>0, stop.

Where Γ is a threshold value, which defines how much frequency deviation of two N-grams is acceptable. It is kept either zero or to a minimum value. Recursion step does the looping through the increasing N-grams of the input word. If at the end of Phase 1, $\psi_N = |W|$, move to

Phase 2:

If the frequencies of last three N-grams (F_i s) are identical, delete the last three characters of the input word only if deleting them makes the resulting stem greater than three, i.e.

$$\psi_N = \psi_N - 3 \quad \text{If} \quad \psi_N - 3 > 3$$

At the end of the above procedure, ψ_N contains the value of the N-gram that approximates the required stem. Phase 2 controls over that words that exhibit common suffixes (like *–ing*), if they aren't stripped in Phase 1.

3. Illustrations and Experiments:

Let us apply our method to the cluster of English, Spanish and Portuguese words. What we need is sequential frequencies of N-grams from a rich corpus. For English language, we relied on COCA . This corpus has two benefits, first it has a rich collection of English words and second, it is easy to calculate the N-gram frequencies using wild card. Sequential frequencies for Spanish and Portuguese words are taken from Corpus Del Español and Corpus Do Português respectively. The results over some testing candidates are being tabled under:

Table 3: Word clusters and stems

Cluster no.	Word	Stem	Cluster no.	Word	Stem
1 (English)	create	creat	2 (Spanish)	trabajan	traba
	creates	creat		trabajar	traba
	creating	creat		trabajado	traba
	created	creat		trabajador	traba
	creation	creat	3 (Portuguese)	dificil	dific
	creative	creat		dificilmente	dific

Now, let us compare our stemmer empirically with an established linguistic stemmer, say Porter's stemmer . We select English as our testing language as outputs of Porter's algorithm can be easily extracted at Snowball project . We take sample of 100 random English words and apply Porter's Snowball framework and our procedure over them. The results are reported in Table 3.2. Identical stems generated by either algorithm are shown in bold. Four POS, Noun, Adjective, Verb and Adverb were taken into account. We select *Newspaper* search area in COCA.

To compare the stemming results of Snowball (Porter) stemmer and our N-gram stemmer, we employed direct evaluation method. Like the comparison made by Chris D. Paise , using length truncation as a baseline, we employed *Levenshtein distance* as a base measure. The distance is the number of deletions, insertions, or substitutions required to transform the source string into the target string. The Levenshtein distance therefore, represents by how many units a word has been stripped by a stemmer. For our sample of the 100 words, these distances are shown in Table 3.2, LD_{Snow} and $LD_{N\text{-}gram}$ for Snowball stems and our N-gram stems respectively.

Table 4: Random Words and their stems

S. No.	**Word**	**Snowball stem**	**N-gram stem**	**LDSnow**	**LDN-gram**
1	Parsons	**Parson**	**Parson**	1	1
2	Dilution	**Dilut**	**Dilut**	3	3
3	Agreement	Agreement	Agree	0	4
4	Passion	**Passion**	**Passion**	0	0
5	Cutter	**Cutter**	**Cutter**	0	0
6	Refill	Refill	Refil	0	1
7	Museums	**Museum**	**Museum**	1	1
8	Stallion	**Stallion**	**Stallion**	0	0
9	Braid	Braid	Brai	0	1
10	Fleet	Fleet	Flee	0	1
11	Midwife	Midwif	Midwife	1	0
12	Haze	**Haze**	**Haze**	0	0
13	Prophet	Prophet	Prophe	0	1
14	Vacancy	**Vacanc**	**Vacanc**	1	1
15	Ninety	Nineti	Ninet	1	1
16	Menace	**Menac**	**Menac**	1	1
17	Laceration	Lacer	Lacerat	5	3
18	Training	**Train**	**Train**	3	3
19	Librarian	Librarian	Librar	0	3
20	Ordaining	**Ordain**	**Ordain**	3	3
21	Mainstream	Mainstream	Mainst	0	4

22	Bloodier	Bloodier	Blood	0	3
23	Abject	**Abject**	**Abject**	0	0
24	Substitute	**Substitut**	**Substitut**	1	1
25	Overseas	Oversea	Overseas	1	0
26	Freehand	Freehand	Freehan	0	1
27	Despotic	Despot	Despoti	2	1
28	Predicted	Predict	Predic	2	3
29	Lyric	**Lyric**	**Lyric**	0	0
30	Eleventh	Eleventh	Eleven	0	2
31	Admonishing	Admonish	Admoni	3	5
32	Quiet	**Quiet**	**Quiet**	0	0
33	Macabre	Macabr	Maca	1	3
34	Unloaded	**Unload**	**Unload**	2	2
35	Quizzical	Quizzic	Quizzi	2	3
36	Alto	**Alto**	**Alto**	0	0
37	Undeveloped	Undevelop	Undevelo	2	3
38	Casual	**Casual**	**Casual**	0	0
39	Recognized	Recogn	Recogni	4	3
40	Devastating	Devast	Devastat	5	3
41	Abused	Abus	Abuse	2	1
42	Abusing	Abus	Abusing	3	0
43	Admired	Admir	Admire	2	1
44	Admiring	**Admir**	**Admir**	3	3
45	Believed	Believ	Belie	2	3
46	Believing	Believ	Belie	3	4
47	Borrowed	**Borrow**	**Borrow**	2	2
48	Borrowing	**Borrow**	**Borrow**	3	3
49	Carried	Carri	Carrie	2	1
50	Carrying	Carri	Carry	3	3
51	Consulted	**Consult**	**Consult**	2	2

52	Consulting	**Consult**	**Consult**	3	3
53	Deceived	Deceiv	Deceive	2	1
54	Deceiving	**Deceiv**	**Deceiv**	3	3
55	Decorated	**Décor**	**Decor**	4	4
56	Decorating	**Décor**	**Decor**	5	5
57	Employed	**Employ**	**Employ**	2	2
58	Employing	**Employ**	**Employ**	3	3
59	Explained	**Explain**	**Explain**	2	2
60	Explaining	**Explain**	**Explain**	3	3
61	Finished	**Finish**	**Finish**	2	2
62	Finishing	**Finish**	**Finish**	3	3
63	Forbidden	Forbidden	Forbid	0	3
64	Forbidding	**Forbid**	**Forbid**	4	4
65	Gathered	**Gather**	**Gather**	2	2
66	Gathering	**Gather**	**Gather**	3	3
67	Improved	**Improv**	**Improv**	2	2
68	Improving	**Improv**	**Improv**	3	3
69	Laughed	**Laugh**	**Laugh**	2	2
70	Laughing	**Laugh**	**Laugh**	3	3
71	Listened	**Listen**	**Listen**	2	2
72	Listening	**Listen**	**Listen**	3	3
73	Mended	Mend	Mende	2	1
74	Mending	Mend	Mending	3	0
75	Nipped	Nip	Nippe	3	1
76	Nipping	Nip	Nipp	4	3
77	Plucked	**Pluck**	**Pluck**	2	2
78	Plucking	**Pluck**	**Pluck**	3	3
79	Preached	**Preach**	**Preach**	2	2
80	Preaching	**Preach**	**Preach**	3	3
81	Enormously	**Enorm**	**Enorm**	5	5
82	Monthly	**Month**	**Month**	2	2

83	Solemnly	**Solemn**	**Solemn**	2	2
84	Abnormally	Abnorm	Abnormal	4	2
85	Diligently	Dilig	Diligen	5	3
86	Jubilantly	**Jubil**	**Jubil**	5	5
87	Frightfully	**Fright**	**Fright**	5	5
88	Swiftly	**Swift**	**Swift**	2	2
89	Miserable	**Miser**	**Miser**	4	4
90	Thankfully	Thank	Thankful	5	2
91	Blissfully	Bliss	Blissful	5	2
92	Reluctantly	Reluct	Reluctan	5	3
93	Viciously	**Vicious**	**Vicious**	2	2
94	Wonderfully	**Wonder**	**Wonder**	5	5
95	Hopelessly	**Hopeless**	**Hopeless**	2	2
96	Briskly	Brisk	Briskly	2	0
97	Delightfully	**Delight**	**Delight**	5	5
98	Anxiously	**Anxious**	**Anxious**	2	2
99	Obnoxiously	Obnoxi	Obnoxious	5	2
100	Inwardly	**Inward**	**Inward**	2	2

We hypothesize that if our stemmer is as efficient as that of Porter's, then the distributions of Levenshtein distances between a word and its stripped stem tend to be same. For a given word, we have two treatments in hand, Porter's algorithm and our N-gram procedure. Thus for a given word, we have a pair of LDs. For comparison, we set the null hypothesis H_0 that no differences exist between the two stemmers.

Because the two sets of measures can be considered as two measures associated to the same sample, we decide to employ a statistical test for paired samples. In particular, a nonparametric statistical test, the *Wilcoxon signed-rank test* is employed by us, since there was no evidence about the distribution of these distances. The Wilcoxon test is based on two paired series (x_i, y_i)

of *N* observed values, one series for each out of the observed variables *X*, *Y* to be compared. The absolute differences $d_i=abs\ (x_i-y_i)$ and $sign\ (d_i)$ of differences are to be computed next. Then the absolute values are ranked, discarding zeros. If N_r is the reduced sample size, the final test statistic is $W =|\sum^{N\ r}[sign(d_i) * rank\ (|d_i|)]|$, which is approximated by a Normal variable for large N.

4. Experimental Results:

On conducting the Wilcoxon test over our sample data, we get the *p* value of 0.54. As $p>0.05$, fail to reject the null hypothesis H_0. Thus, we have strong evidence that our N-gram Stemmer is not inferior to Porter's Stemmer.

Naïve bayes classifier :

A **Bayes classifier** is a simple probabilistic classifier based on applying Bayes' theorem (from Bayesian statistics) with strong (naive) independence assumptions. A more descriptive term for the underlying probability model would be "independentfeature model".

In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 4" in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier considers all of these properties to independently contribute to the probability that this fruit is an apple.

Depending on the precise nature of the probability model, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method ofmaximum likelihood; in other words, one can work with the naive Bayes model without believing in Bayesian probability or using any Bayesian methods.

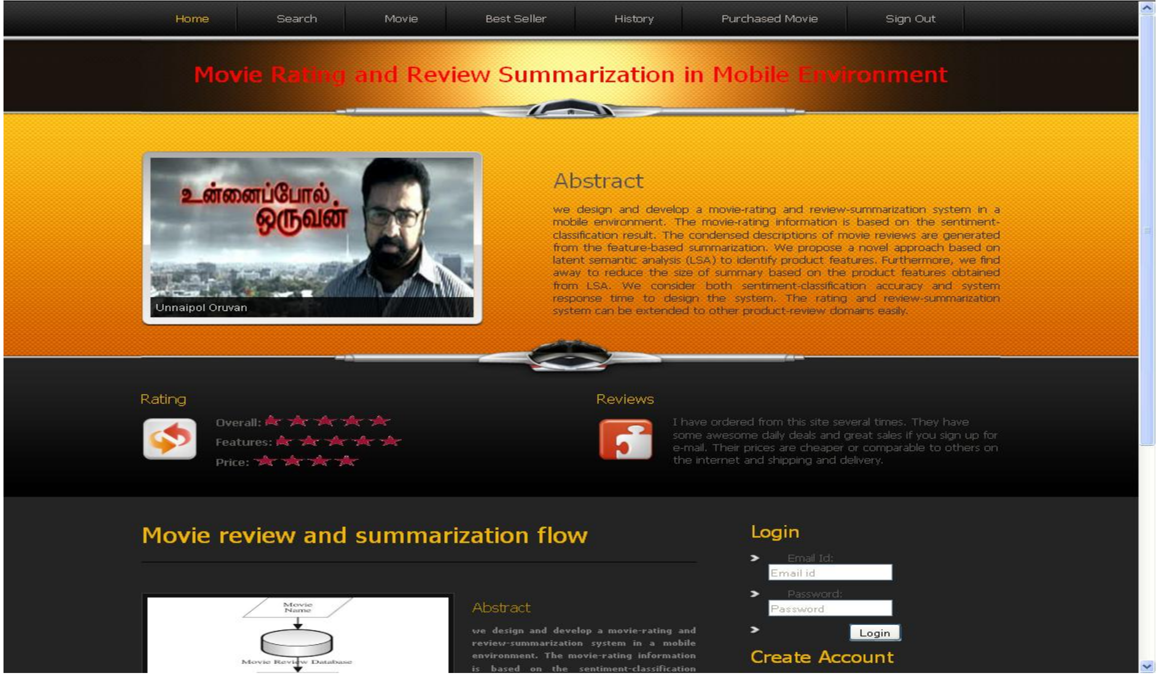
In spite of their naive design and apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. In 2004, analysis of the Bayesian classification problem has shown that there are some theoretical reasons for the apparently unreasonable efficacy of naive Bayes classifiers.[1] Still, a comprehensive comparison with other classification methods in 2006 showed that Bayes classification is outperformed by more current approaches, such as boosted trees or random forests.[2]

An advantage of the naive Bayes classifier is that it requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification.

Because independent variables are assumed, only the variances of the variables for each class need to be determined and not the entirecovariance matix.

RESULTS:

Home page



Movie Rating and Review Summarization in Mobile Environment

Abstract

we design and develop a movie-rating and review-summarization system in a mobile environment. The movie-rating information is based on the sentiment-classification result. The condensed descriptions of movie reviews are generated from the feature-based summarization. We propose a novel approach based on latent semantic analysis (LSA) to identify product features. Furthermore, we find away to reduce the size of summary based on the product features obtained from LSA. We consider both sentiment-classification accuracy and system response time to design the system. The rating and review-summarization system can be extended to other product-review domains easily.

Rating

Overall: ★ ★ ★ ★ ★

Features: ★ ★ ★ ★ ★

Price: ★ ★ ★ ★

Reviews

I have ordered from this site several times. They have some awesome daily deals and great sales if you sign up for e-mail. Their prices are cheaper or comparable to others on the internet and shipping and delivery.

Movie review and summarization flow

Abstract

we design and develop a movie-rating and review-summarization system in a mobile environment. The movie-rating information is based on the sentiment-classification

Login

Email Id: Email id

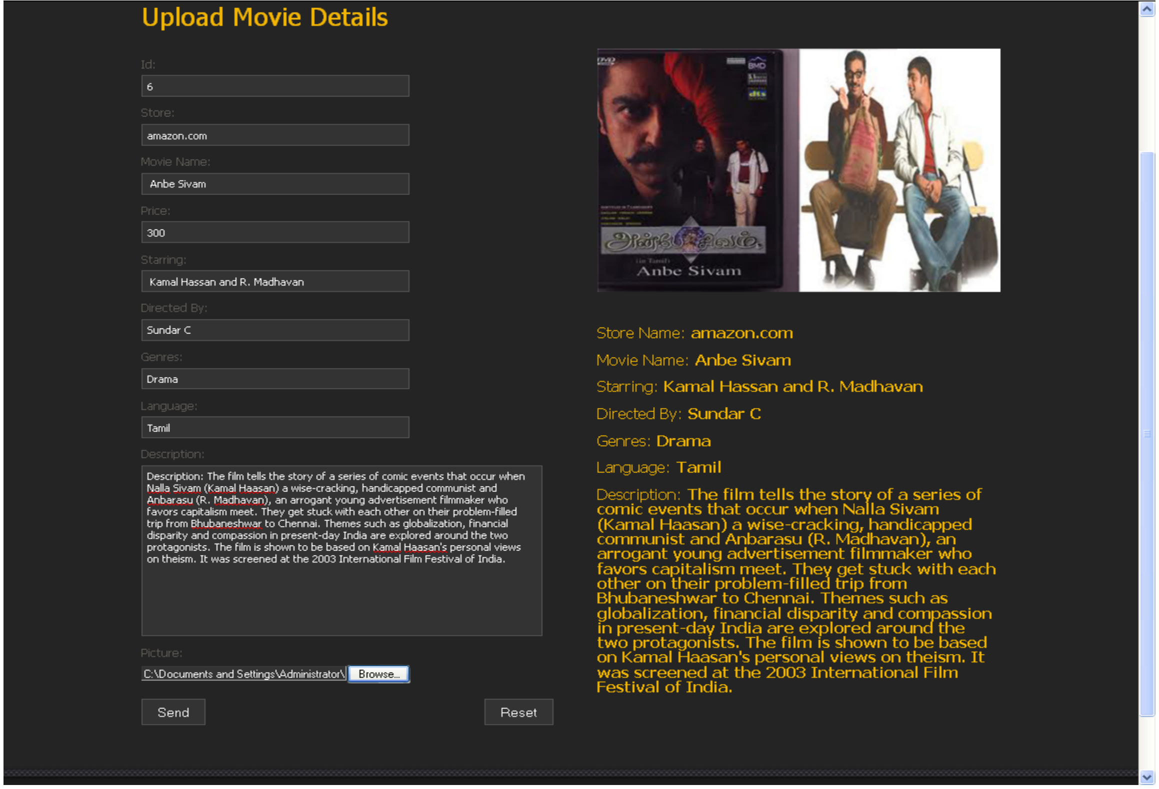
Password: Password

Login

Create Account

Admin Page

Upload Movie Details



Upload Movie Details

Id:
6

Store:
amazon.com

Movie Name:
Anbe Sivam

Price:
300

Starring:
Kamal Hassan and R. Madhavan

Directed By:
Sundar C

Genres:
Drama

Language:
Tamil

Description:
Description: The film tells the story of a series of comic events that occur when Nalla Sivam (Kamal Haasan) a wise-cracking, handicapped communist and Anbarasu (R. Madhavan), an arrogant young advertisement filmmaker who favors capitalism meet. They get stuck with each other on their problem-filled trip from Bhubaneshwar to Chennai. Themes such as globalization, financial disparity and compassion in present-day India are explored around the two protagonists. The film is shown to be based on Kamal Haasan's personal views on theism. It was screened at the 2003 International Film Festival of India.

Picture:
C:\Documents and Settings\Administrator\

Store Name: amazon.com

Movie Name: Anbe Sivam

Starring: Kamal Hassan and R. Madhavan

Directed By: Sundar C

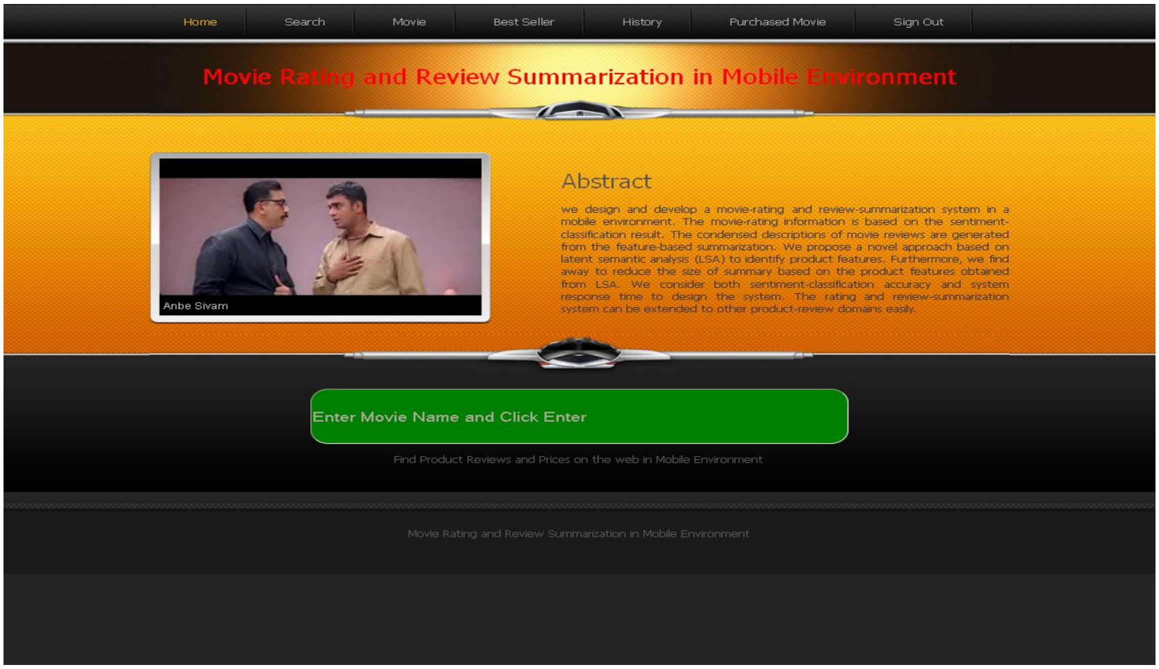
Genres: Drama

Language: Tamil

Description: The film tells the story of a series of comic events that occur when Nalla Sivam (Kamal Haasan) a wise-cracking, handicapped communist and Anbarasu (R. Madhavan), an arrogant young advertisement filmmaker who favors capitalism meet. They get stuck with each other on their problem-filled trip from Bhubaneshwar to Chennai. Themes such as globalization, financial disparity and compassion in present-day India are explored around the two protagonists. The film is shown to be based on Kamal Haasan's personal views on theism. It was screened at the 2003 International Film Festival of India.

User Page

Search Movie



Movie Rating and Review Summarization in Mobile Environment

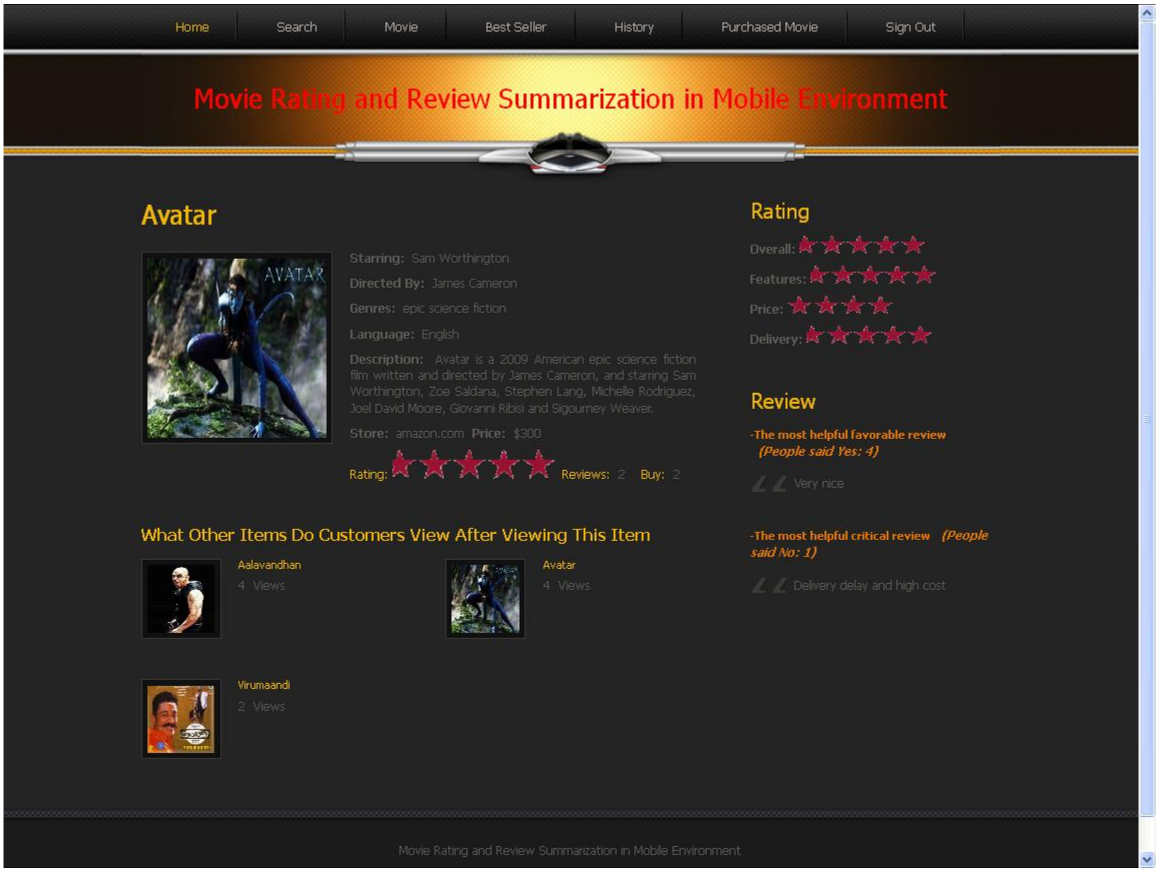
Abstract

we design and develop a movie-rating and review-summarization system in a mobile environment. The movie-rating information is based on the sentiment-classification result. The condensed descriptions of movie reviews are generated from the feature-based summarization. We propose a novel approach based on latent semantic analysis (LSA) to identify product features. Furthermore, we find away to reduce the size of summary based on the product features obtained from LSA. We consider both sentiment-classification accuracy and system response time to design the system. The rating and review-summarization system can be extended to other product-review domains easily.

Find Product Reviews and Prices on the web in Mobile Environment

Movie Rating and Review Summarization in Mobile Environment

View Movie Details



Movie Rating and Review Summarization in Mobile Environment

Avatar

Starring: Sam Worthington

Directed By: James Cameron

Genres: epic science fiction

Language: English

Description: Avatar is a 2009 American epic science fiction film written and directed by James Cameron, and starring Sam Worthington, Zoe Saldana, Stephen Lang, Michelle Rodriguez, Joel David Moore, Giovanni Ribisi and Sigourney Weaver.

Store: amazon.com Price: $300

Rating: ★★★★★ Reviews: 2 Buy: 2

Rating

Overall: ★★★★★

Features: ★★★★★

Price: ★★★★

Delivery: ★★★★★

Review

-The most helpful favorable review (People said Yes: 4)

Very nice

-The most helpful critical review (People said No: 1)

Delivery delay and high cost

What Other Items Do Customers View After Viewing This Item

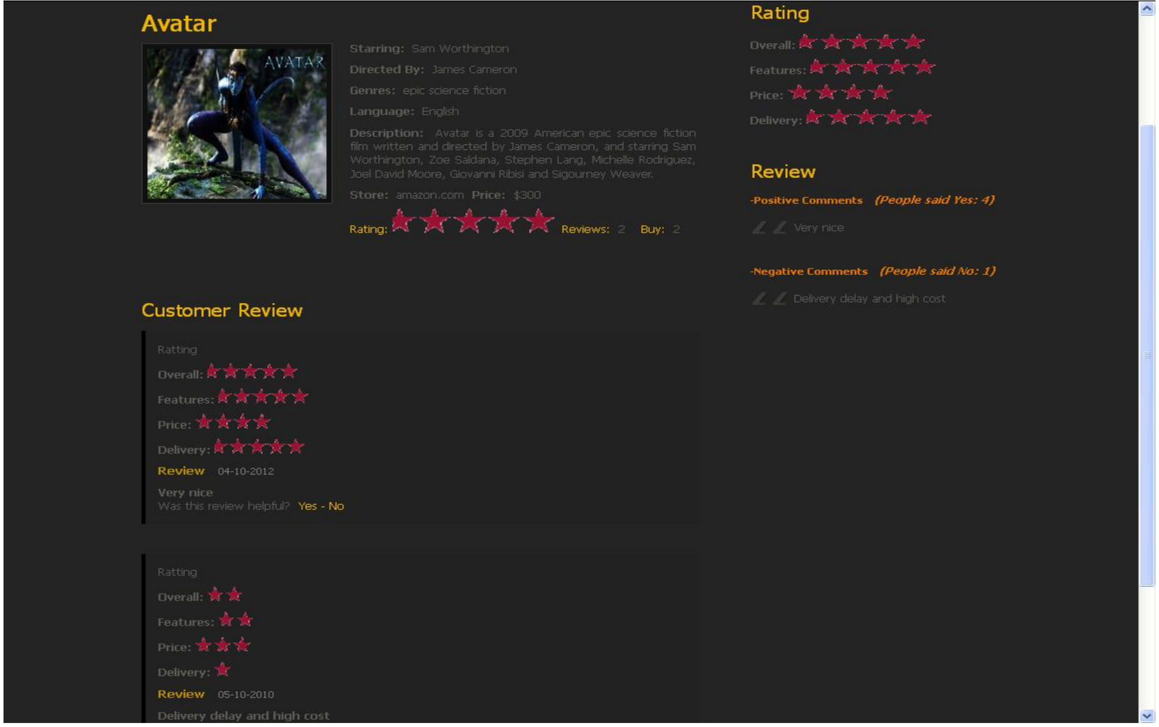
Aalavandhan
4 Views

Avatar
4 Views

Virumaandi
2 Views

Movie Rating and Review Summarization in Mobile Environment

View Rating and Review Summ



Avatar

Starring: Sam Worthington

Directed By: James Cameron

Genres: epic science fiction

Language: English

Description: Avatar is a 2009 American epic science fiction film written and directed by James Cameron, and starring Sam Worthington, Zoe Saldana, Stephen Lang, Michelle Rodriguez, Joel David Moore, Giovanni Ribisi and Sigourney Weaver.

Store: amazon.com Price: $300

Rating: ★★★★★ Reviews: 2 Buy: 2

Rating

Overall: ★★★★★

Features: ★★★★★

Price: ★★★★

Delivery: ★★★★★

Review

-Positive Comments (People said Yes: 4)

Very nice

-Negative Comments (People said No: 1)

Delivery delay and high cost

Customer Review

Ratting

Overall: ★★★★★

Features: ★★★★★

Price: ★★★★

Delivery: ★★★★★

Review 04-10-2012

Very nice

Was this review helpful? Yes - No

Ratting

Overall: ★★

Features: ★★

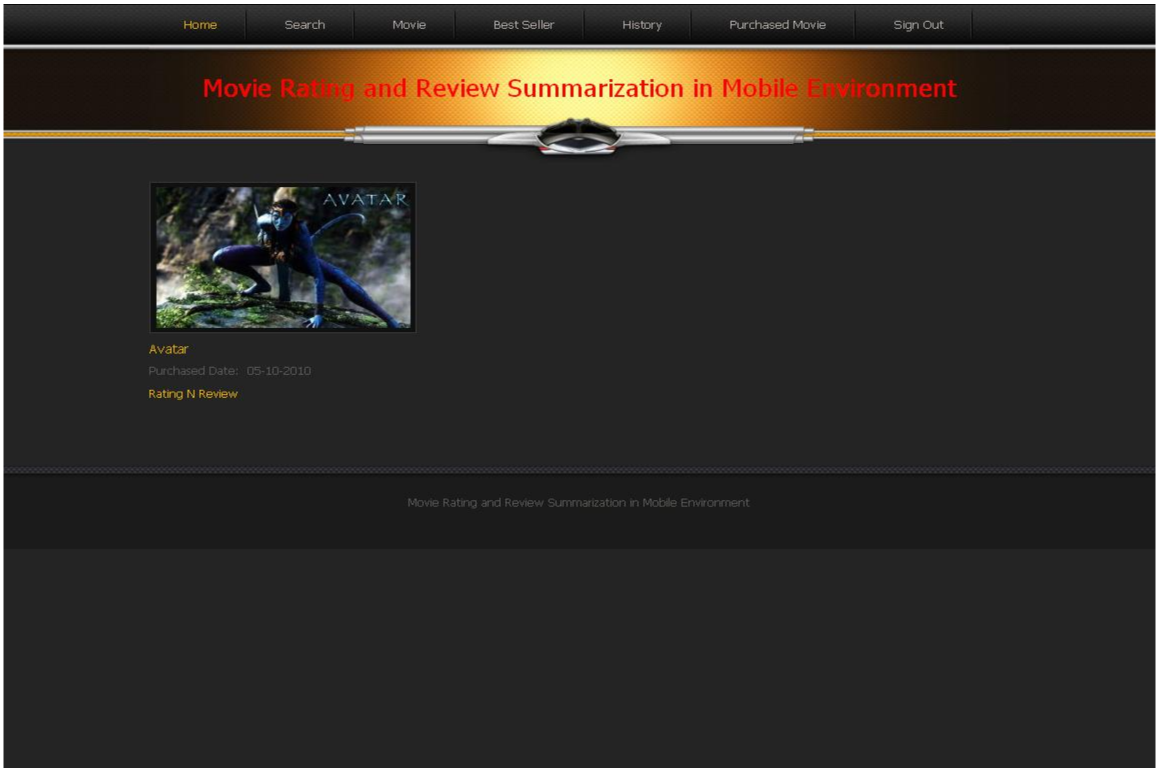
Price: ★★★

Delivery: ★

Review 05-10-2010

Delivery delay and high cost

Purchase Movie



Movie Rating and Review Summarization in Mobile Environment

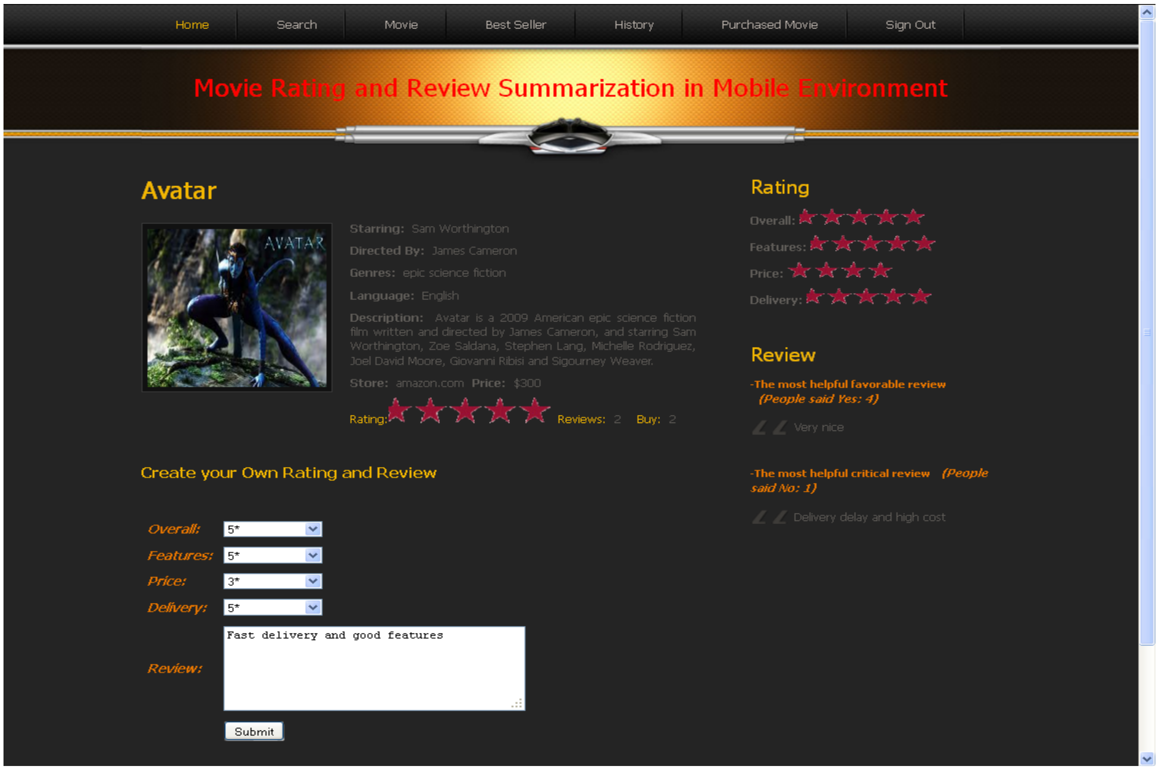
Avatar

Purchased Date: 05-10-2010

Rating N Review

Movie Rating and Review Summarization in Mobile Environment

Customer Review



Movie Rating and Review Summarization in Mobile Environment

Avatar

Starring: Sam Worthington

Directed By: James Cameron

Genres: epic science fiction

Language: English

Description: Avatar is a 2009 American epic science fiction film written and directed by James Cameron, and starring Sam Worthington, Zoe Saldana, Stephen Lang, Michelle Rodriguez, Joel David Moore, Giovanni Ribisi and Sigourney Weaver.

Store: amazon.com **Price:** $300

Rating: ★★★★★ Reviews: 2 Buy: 2

Rating

Overall: ★★★★★

Features: ★★★★★

Price: ★★★★

Delivery: ★★★★★

Review

-The most helpful favorable review *(People said Yes: 4)*

Very nice

-The most helpful critical review *(People said No: 1)*

Delivery delay and high cost

Create your Own Rating and Review

Overall: 5*

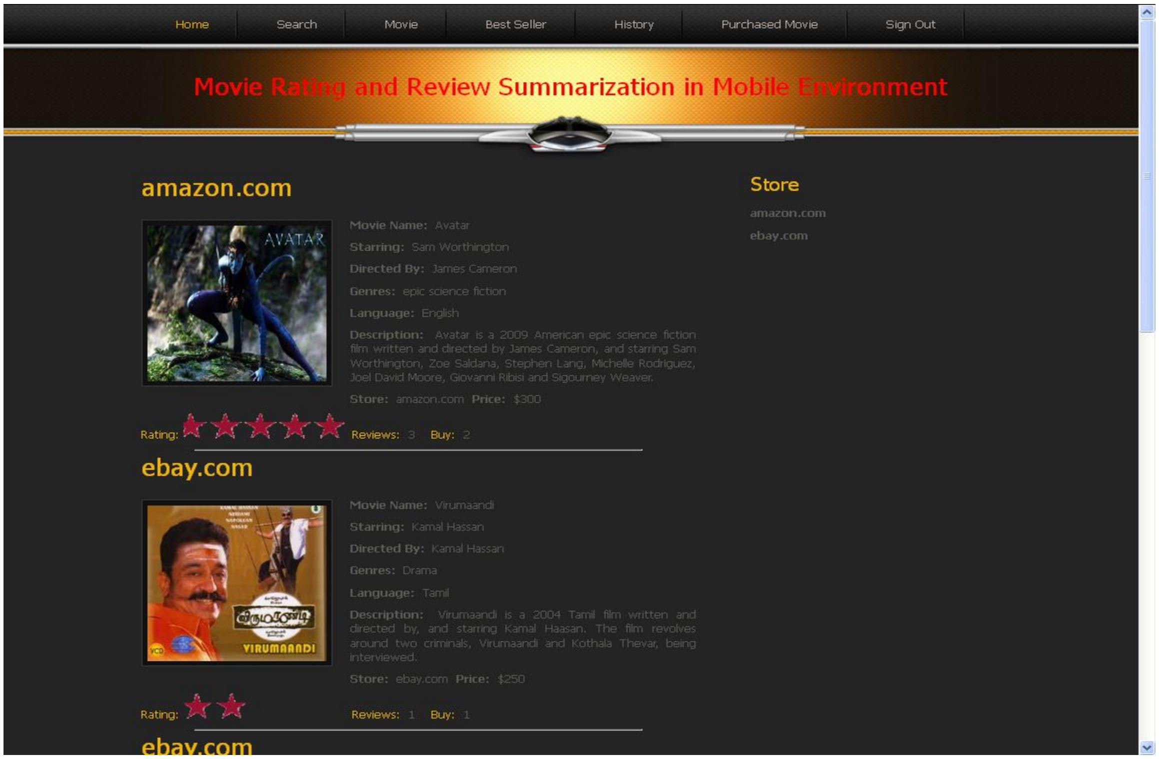
Features: 5*

Price: 3*

Delivery: 5*

Review: Fast delivery and good features

Best Seller



amazon.com

Movie Name: Avatar

Starring: Sam Worthington

Directed By: James Cameron

Genres: epic science fiction

Language: English

Description: Avatar is a 2009 American epic science fiction film written and directed by James Cameron, and starring Sam Worthington, Zoe Saldana, Stephen Lang, Michelle Rodriguez, Joel David Moore, Giovanni Ribisi and Sigourney Weaver.

Store: amazon.com Price: $300

Rating: ★★★★★ Reviews: 3 Buy: 2

ebay.com

Movie Name: Virumaandi

Starring: Kamal Hassan

Directed By: Kamal Hassan

Genres: Drama

Language: Tamil

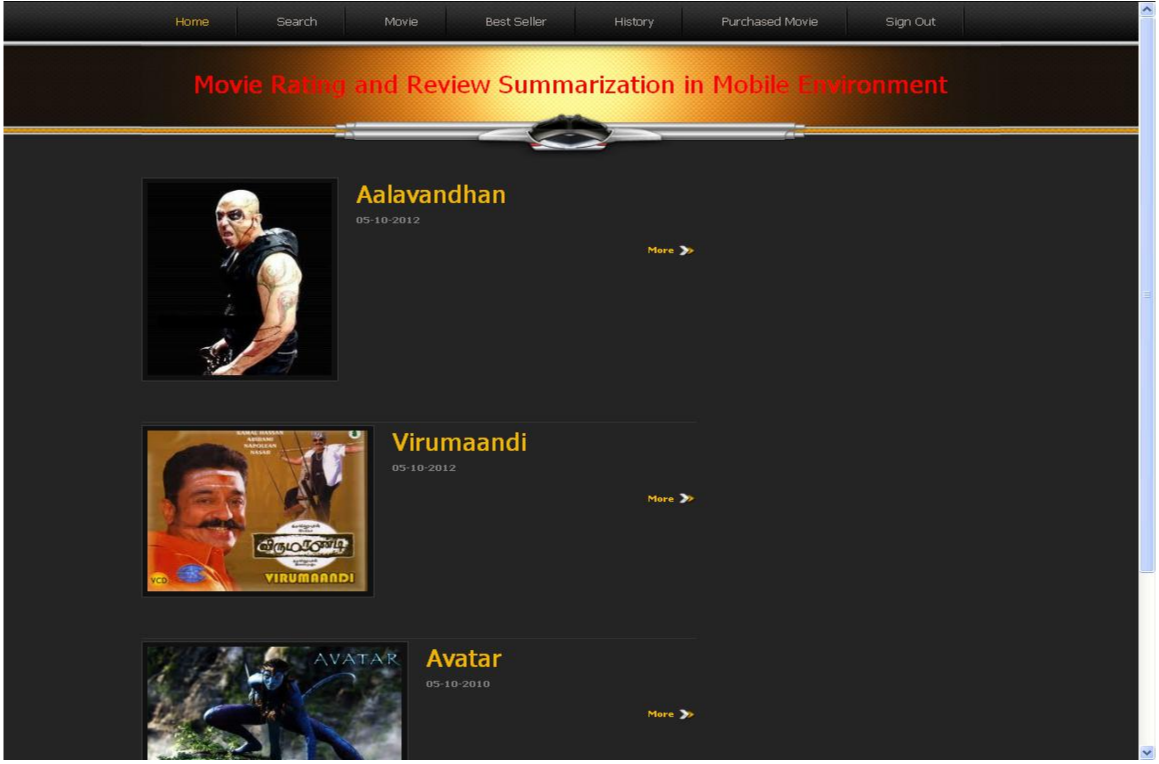
Description: Virumaandi is a 2004 Tamil film written and directed by, and starring Kamal Haasan. The film revolves around two criminals, Virumaandi and Kothala Thevar, being interviewed.

Store: ebay.com Price: $250

Rating: ★★ Reviews: 1 Buy: 1

ebay.com

Recent History



Aalavandhan

05-10-2012

Virumaandi

05-10-2012

Avatar

05-10-2010

Conclusions:

We endeavored to modify a statistical stemming technique and came up with one whose results are comparable with a well known linguistic stemmer. Our N-gram stemmer is capable to deal any language for which N-gram corpus frequencies are available. As a language neutral approach is always preferable over a technique in which a linguistic knowledge or analysis is prerequisite, our results seem to be very exciting and promising.

It is really very important ant to identify the factors that improve the performances as well .factors that affect performance are the choice of training data, attribute or feature selection, representation of instances ,and the choice of the algorithm used. A decision to make equal number of each class that is positive and negative was being made to avoid biasing of classifier in favor of any particular class.

Attribute selection effected the accuracy that we can get from the classifier .too few attributes can cause the under fitting and too many attributes causing over fitting.

Under fitting s a situation in which the classifier is not biased enough to make prediction on some unseen instances. Over fitting is a situation where the train model is highly fitted to the training data that it basically fails while predicting new instances successfully. Thus it s good to find a balance between too few and too many attributes .unfortunately, there seems to be no as such simple way of finding ths balanced number of attributes except by trail and error.

The other factors that affected the performance is the representation of instances .this basically includes whether to use presence /absence or count/frequency.it also includes whether to convert count into tf-idf so that terms importance is taken into account. Thae best result for multinomial naïve bayes can be obtained using this presence. Finally, the choice of machine learning algorithm for the task also affected the performance. Multinomial naïve bayes was found to outperform others in the task of sentiment analysis /classification task.

Future Work

It is not possible to develop a system that meets all the requirements of the user. User requirements keep changing as the system is being used. Some of the future enhancements that can be done to this system are:

- As the technology emerges, it is possible to upgrade the system that can be adaptable to desired environment.

Refrences :

1. Zhiyuan Chen, Arjun Mukherjee, Bing Liu, Meichun Hsu, Malu Castellanos, and Riddhiman Ghosh. Exploiting Domain Knowledge in Aspect Extraction. *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP-2013)*, October 18-21, 2013, Seattle, USA.
2. Arjun Mukherjee, Vivek Venkataraman, Bing Liu, and Sharon Meraz. Public Dialogue: Analysis of Tolerance in Online Discussions. *Proceedings of The 51st Annual Meeting of the Association for Computational Linguistics (ACL-2013)*, August 4-9, 2013, Sofia, Bulgaria.
3. Arjun Mukherjee, Bing Liu. Discovering User Interactions in Ideological Discussions. *Proceedings of The 51st Annual Meeting of the Association for Computational Linguistics (ACL-2013)*, August 4-9, 2013, Sofia, Bulgaria.
4. Jianfeng Si, Arjun Mukherjee, Bing Liu, Qing Li, Huayi Li, and Xiaotie Deng. Exploiting Topic based Twitter Sentiment for Stock Prediction. *Proceedings of The 51st Annual Meeting of the Association for Computational Linguistics (ACL-2013, short paper)*, August 4-9, 2013, Sofia, Bulgaria.
5. Zhiyuan Chen, Bing Liu, Meichun Hsu, Malu Castellanos, and Riddhiman Ghosh. Identifying Intention Posts in Discussion Forums. *Proceedings of The 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*, June 9-15, 2013, Atlanta, USA.
6. Bing Liu. Sentiment Analysis and Opinion Mining. Morgan & Claypool Publishers, May 2012.
7. Arjun Mukherjee and Bing Liu. Modeling Review Comments. *Proceedings of 50th Annual Meeting of Association for Computational Linguistics (ACL-2012)*, July 8-14, 2012, Jeju, Republic of Korea.
8. Arjun Mukherjee and Bing Liu. Aspect Extraction through Semi-Supervised Modeling. *Proceedings of 50th Annual Meeting of Association for Computational Linguistics (ACL-2012)*, July 8-14, 2012, Jeju, Republic of Korea.
9. Arjun Mukherjee and Bing Liu. Mining Contentions from Discussions and Debates. *Proceedings of SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2012)*, Aug. 12-16, 2012, Beijing, China.
10. Lei Zhang and Bing Liu. "Extracting Resource Terms for Sentiment Analysis," *Proceedings of the 5th International Joint Conference on Natural Language Processing (IJCNLP-2011)*, November 8-13, 2011, Chiang Mai, Thailand.
11. Zhongwu Zhai, Bing Liu, Lei Zhang, Hua Xu, Peifa Jia. Identifying Evaluative Opinions in Online Discussions. *Proceedings of AAAI-2011*, San Francisco, USA, August 7-11, 2011.
12. Lei Zhang and Bing Liu. "Identifying Noun Product Features that Imply Opinions." *ACL-2011* (short paper), Portland, Oregon, USA, June 19-24, 2011.
13. Guang Qiu, Bing Liu, Jiajun Bu and Chun Chen. "Opinion Word Expansion and Target Extraction through Double Propagation." *Computational Linguistics*, March 2011, Vol. 37, No. 1: 9.27.

14. Zhongwu Zhai, Bing Liu, Hua Xu, Peifa Jia. "Constrained LDA for Grouping Product Features in Opinion Mining." *Proceedings of PAKDD-2011*, Shenzhen, China, 2011. (***Best Paper Award***)
15. Lei Zhang and Bing Liu. "Entity Set Expansion in Opinion Documents." *Proceedings of the ACM Conference on Hypertext and Hypermedia* (HT-2011), Eindhoven, Netherlands, June 6-9, 2011.
16. Zhongwu Zhai, Bing Liu, Hua Xu and Peifa Jia. "Clustering Product Features for Opinion Mining." *Proceedings of Fourth ACM International Conference on Web Search and Data Mining (WSDM-2011)*, Feb. 9-12, 2011, Hong Kong, China.
17. Arjun Mukherjee and Bing Liu. "Improving Gender Classification of Blog Authors." *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP-10)*. Oct. 9-11, 2010, MIT, Massachusetts, USA.
18. Xiaowen Ding and Bing Liu. "Resolving Object and Attribute Coreference in Opinion Mining." *Proceedings of the 23rd International Conference on Computational Linguistics (COLING-2010)*, August 23-27, Beijing, China.
19. Zhongwu Zhai, Bing Liu, Hua Xu and Peifa Jia. "Grouping Product Features Using Semi-Supervised Learning with Soft-Constraints" *Proceedings of the 23rd International Conference on Computational Linguistics (COLING-2010)*, August 23-27, Beijing, China.
20. Lei Zhang and Bing Liu. "Extracting and Ranking Product Features in Opinion Documents." *Proceedings of the 23rd International Conference on Computational Linguistics (COLING-2010)*, August 23-27, Beijing, China.
21. Bing Liu. "Sentiment Analysis: A Multifaceted Problem." Invited paper, *IEEE Intelligent Systems*, 25(3), 2010, pp. 76-80.
22. Bing Liu. "Sentiment Analysis and Subjectivity." Invited Chapter for the *Handbook of Natural Language Processing*, Second Edition. March, 2010.
23. Ramanathan Narayanan, Bing Liu and Alok Choudhary. "Sentiment Analysis of Conditional Sentences." *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP-09)*. August 6-7, 2009. Singapore.
24. Guang Qiu, Bing Liu, Jiajun Bu and Chun Chen. "Expanding Domain Sentiment Lexicon through Double Propagation." *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, Pasadena, California, USA, July 11-17, 2009.
25. Xiaowen Ding, Bing Liu and Lei Zhang. "Entity Discovery and Assignment for Opinion Mining Applications," *Proceedings of ACM SIGKDD Interntaional Conference on Knowledge Disocvery and Data Mining (KDD-09, industrial track)*, June 28-July 1, 2009, Paris.
26. Bing Liu. "Opinion Mining." Invited contribution to *Encyclopedia of Database Systems*, 2008.
27. Murthy Ganapathibhotla and Bing Liu. "Mining Opinions in Comparative Sentences." *Proceedings of the 22nd International Conference on Computational Linguistics (Coling-2008)*, Manchester, 18-22 August, 2008.
28. Xiaowen Ding, Bing Liu and Philip S. Yu. "A Holistic Lexicon-Based Appraoch to Opinion Mining." *Proceedings of First ACM International Conference on Web Search and Data Mining (WSDM-2008)*, Feb 11-12, 2008, Stanford University, Stanford, California, USA.

29. Xiaowen Ding and Bing Liu. "The Utility of Linguistic Rules in Opinion Mining." *SIGIR-2007* (poster paper), 23-27 July 2007, Amsterdam.
30. Nitin Jindal and Bing Liu. "Identifying Comparative Sentences in Text Documents" *Proceedings of the 29th Annual International ACM SIGIR Conference on Research & Development on Information Retrieval (SIGIR-06)*, Seattle 2006.
31. Nitin Jindal and Bing Liu. "Mining Comprative Sentences and Relations." *Proceedings of 21st National Conference on Artificial Intellgience (AAAI-2006)*, July 16.20, 2006, Boston, Massachusetts, USA.
32. Bing Liu, Minqing Hu and Junsheng Cheng. "Opinion Observer: Analyzing and Comparing Opinions on the Web"*Proceedings of the 14th international World Wide Web conference (WWW-2005)*, May 10-14, 2005, in Chiba, Japan.
33. Minqing Hu and Bing Liu. "Mining Opinion Features in Customer Reviews." *Proceedings of Nineteeth National Conference on Artificial Intellgience (AAAI-2004)*, San Jose, USA, July 2004.
34. Minqing Hu and Bing Liu. "Mining and summarizing customer reviews." *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2004, full paper)*, Seattle, Washington, USA, Aug 22-25, 2004.

i