MovieOracle System

(Recommender System on Twitter)

Detailed Design Document

Author: Yao Yao

TABLE OF CONTENTS

1. Introduction

The detailed design document covers the basic theory, related technique, and implementation details of the MovieOracle system. It first provides a high-level system overview. Then it discusses the usage of Twitter API, Twitter4J and CI-Bayes. After that, it describes how components communicate with each other, how they are organized, and how each of them is implemented. Finally the document introduces the sentiment analysis and the decision tree method of the project. A program implementation for this project should follow the direction of this document.

1.1 References and Related Documents

Document	Author
Software Requirements Specification (MovieOracle System)	Yao Yao
MovieOracle User Tutorial	Yao Yao

2. System Overview

This project provides the service of recommending movies to appropriate twitter users. It is implemented in Java and is supported by MySQL database. The system specification is in the Software Requirements Specification document.

2.1 Functional Requirements Overview

The major functions of this system include: collecting tweet information of specified movies; finding potential users for prediction; classifying tweets; predicting twitter users' opinions on movies. It also provides a command line interface and a log service.

2.2 High Level Component Model

Besides a database, this project is composed of five major components: Tweet Collection, Potential User Finder, Polarity Classifier, Decision Tree and System Service. Each component corresponds to different functional requirements as follows.

1. Tweet Collection: collecting tweet information of specified movies
2. Potential User Finder: finding potential users for prediction
3. Polarity Classifier: classifying tweets into three categories
4. Decision Tree: predicting twitter users' opinions on movies
5. System Service: command line interface and log service

2.2.1 Component Descriptions

1. Tweet Collection

Tweet Collection is used to collect tweet information of specified movies and store them in the database. The Twitter Search API (see Section 3) limits calling its search method within each hour and every search returns the 100 latest tweets containing the key word or phrase (movie name).

2. Polarity Classifier

Polarity Classifier periodically classifies tweets' sentiments into three categories: positive, negative, and unknown. The time interval of periodical classification is determined by the MovieOracle user. It writes the classification results to the database.

3. Potential User Finder

Potential User Finder periodically calls methods of the Twitter Rest API (see Section 3) to get tweet authors' friends and followers. It analyzes tweet information in the database to find the potential users, which are users in the system with sufficient related tweets[1] to qualify the decision tree prediction. After that, it invokes the decision tree to predict potential users' opinions on specified movies.

4. Decision Tree

The decision tree is constructed after the MovieOracle system starts. (This allows MovieOracle users to use their own training data to construct the decision tree.) It is then invoked by the Potential User Finder component to predict the potential users' sentiments. The attributes used in the tree are percentages of positive, negative, and unknown related tweets of a potential user.

5. System Service

System Service invokes and coordinates some other components. It also provides a log service and a command line interface for the entire system. The log service records in a log file the starting of each component, important operations, warning and error messages. The command line interface accepts commands to manipulate the MovieOracle system. The command specification is introduced in Chapter 4 "Commands" of the MovieOracle User Tutorial document.

----

[1] For Twitter users who publish tweets about some topic, if their friends and followers publish the tweets discussing the same topic, these tweets from their friends and followers are called related tweets of the users.

3. Twitter API [1]

An application can access Twitter data via the Twitter API. Twitter4J [2] is a Java library for the Twitter API. It is applied in the Tweet Collection and Potential User Finder components of this project. The Twitter API (Twitter4J) has its own specification and limitation for the programs accessing it.

3.1 Twitter4J Specification

The Twitter4J introduction can be found on http://twitter4j.org. The major classes and interfaces used in this project are: TwitterFactory, Twitter, Configuration, ConfigurationBuilder, Query, QueryResult, and Tweet. Some of their common methods and usage are discussed as follows.

1. TwitterFactory
   TwitterFactory is a factory class for Twitter. Its constructor used in the project is "TwitterFactory(Configuration conf)". Its "getInstance()" method is used to generate an authenticated Twitter instance. The class of this instance represents the Twitter API.

2. Twitter
   An instance of the Twitter class is generated by the TwitterFactory. In this project, all usage of the Twitter API are accessed via such an instance. Its "search(Query query)" method searches tweets by a key word or phrase. This method is used in the Tweet Collection component. Its "getFollowersIDs(String screenName)" method returns the follower ids of a given screen name. Its "getFriendsIDs(String screenName)" method returns the friend ids of a given screen name. Its "getRateLimitStatus()" method returns an instance of RateLimitStatus. The RateLimitStatus has a method "getRemainingHits()" that returns the remaining number of Twitter REST API requests available. These four methods are used in the Potential User Finder component.

3. ConfigurationBuilder
   ConfigurationBuilder is used to set Twitter API parameters. In this project, these parameters include Token, TokenSecret, ConsumerKey, and ConsumerSecret [3]. These four parameters are provided by Twitter once a twitter-based application is registered. For this project, these parameters are in a property file (see Chapter 5 of the MovieOracle User Tutorial document for detail). Its "build()" method creates an instance of the class which implements Configuration interface.

4. Query
   Query is a class used to search tweets. Its instance is passed to the "search(Query query)" method of a Twitter instance in order to search tweets having the desired

movie names. The parameter in its constructor “Query(String key)” is the desired movie name. Its “setRpp(int rpp)” method sets the number of latest tweets to return per query, up to a max of approximately 100. Its “setLang(String lang)” method restricts return tweets in the given language.

5. QueryResult
   QueryResult is an interface representing the search response. It is implemented by the class of the returned instance of the method “search(Query query)”, which is mentioned in item 2 and 4 in this list. The “getTweets()” method returns a list of Tweet instances. The maximum list size is approximately 100 (limited by the Twitter API).

6. Tweet
   Tweet is a class representing a tweet in the search result. Its “getId()” method returns a tweet status id. The “getFromUserId()” method returns the tweet author id. The “getFromUser()” method returns the tweet author name. The “getText()” method returns the actual tweet.

3.2 Twitter API Limitations

The Twitter API only allows a client to make a limited number of method calls within an hour. More information is available on http://dev.twitter.com/pages/rate-limiting. In this project, retrieving friends or followers is limited by the Rest API rate limit. Authorized applications are permitted 350 requests per hour. The search method is limited by the Search API rate limit. The access rate is not made public but is close to the rate of Rest API rate limit. In this project, the default value of the Search API rate limit is set to 300 (it can be modified to a lower value in the property file).

When using a movie name as the key word to collect tweets, there will be an “unrelated tweet” problem, which means the collected tweets are not all related to the discussion of a desired movie. For example, consider the movie “Red”. A tweet containing the word “Red” could be about the famous fairy tale “Little Red Riding Hood”, or even just some body talked the house painting. One helpful method is to add a string “movie” plus a blank space as the prefix of the movie name while using “start” or “collect” command. Although it can avoid gathering meaningless tweets, it also may miss some valuable tweets that do discuss the movie but have no such prefix. In addition, some tweets contain the movie names which are commonly thought to be used for movies, but they may still talk about something else. For example, “Harry Potter” can be used for a book. When opportunities of such “faulty collecting” are relatively few, it is unnecessary to add the string “movie” to the movie name.

In the future, if new-found methods can well distinguish the tweets about the desired movie from other meaningless ones, the problem discussed in the paragraph above can be solved by modifying code for the tweet collection thread. After the thread successfully retrieves tweets, instead of directly storing them in the database, add the desired method to filter out the unrelated tweets and then only store the meaningful ones (see more details about the design of the tweet collection thread in Section 5.2.4).

4. CI-Bayes[4]

CI-Bayes provides the Fisher classification method for this project. Its basic information and fundamental theory are introduced in Section 6.1.2, 6.1.3, and 6.1.4.

4.1 CI-Bayes Specification

The major class and interface used in this project are: FisherClassifier and FisherClassifierImpl. FisherClassifierImpl implements the interface FisherClassifier. Its common methods are:

1. train(Object tweet, String category)
   It tells the fisher classifier that the “tweet” belongs to the “category”. Given a large number of tweets, this method trains the classifier. In this project, three categories are used: positive, negative, and unknown.

2. getFisherProbability(Object tweet, String category)
   It returns a double number and this value represents the probability that the “tweet” belongs to the “category”.

3. getClassification(Object tweet)
   It returns a string and this value represents the category to which the “tweet” belongs.

4.2 CI-Bayes Usage

Although CI-Bayes has a “getClassification(Object tweet)” method which allow directly classifying a tweet, sometimes the project does not use it to classify tweets. Instead, some simple rules are applied based on the “getFisherProbability(Object tweet, String category)” method. These rules are derived from long-term observation on many experiments for validating the correctness of “getClassification(Object tweet)”. With the “getFisherProbability(Object tweet, String category)” method, let P be the probability of a tweet is “positive”, N be the probability of a tweet is “negative”, U be the probability of a tweet is “unknown”,

1. If U is no greater than both P and N, and the absolute value of $(P - N) < 0.1$, then the tweet is considered “unknown”.
2. If $P <= U <= N$ and $(N - U) < 0.1$, then the tweet is considered “unknown”.
3. If $N <= U <= P$ and $(P - U) < 0.1$, then the tweet is considered “unknown”.
4. If the three rules above fail, then the tweet is directly classified by the “getClassification(Object tweet)” method.

In the future, if new rules are found or the original rules need to be modified, one can implement them by modifying the method which applies rules to classify a tweet in the polarity classifier (see more details about the design of the polarity classifier in section 5.3.4).

5. Application Design

This section introduces the system property file and the detailed design of five major components: Tweet Collection, Potential User Finder, Polarity Classifier, Decision Tree, and System Service. Each subsection describes the component's functional requirements, data model, variables and constants, and algorithm implementation. Since the design of the command line interface has much information, it is discussed in a separate subsection.

5.1 Property File

The MovieOracle system uses a standard property file called "config.properties" to indicate environment variables. The file is in the same directory of the system. It has the following variables used by different components.

Variable	Component
database_name	System Service
related_tweets_threshold	Potential User Finder
polarity_training_set	Polarity Classifier
search_limit	Tweet Collection
clear_interval	System Service
check_tweet_interval	Polarity Classifier
token	System Service
token_secret	System Service
consumer_key	System Service
consumer_secret	System Service

More details of the environment variables can be found in Chapter 5 "Property File" of the MovieOracle User Tutorial document.

5.2 Tweet Collection

The implementation of the Tweet Collection component is supported by five tables introduced in 5.2.2 Data Model. They are MOVIE, AUTHOR, TWEET, THREAD_RESOURCE, and THREAD_CALL. The specification of this part is in 3.2 of the Software Requirements Specification document.

5.2.1 Target and Requirements

The Tweet Collection component targets getting the newest tweets on specified movies and storing them (as well as related information) in the database. Since the search rate is limited by the Twitter Search API and each search returns the latest 100 tweets of a movie, each movie search is run in a separate thread. Every thread periodically collects tweets of one movie. It uses primary key constraint of the TWEET table to avoid storing duplicate tweets.

The sleep time of each thread is evenly divided among all threads currently running in the system. Let cph (calls per hour) represent the number of available calls per hour for a thread. Then cph = Twitter search limit rate/the total number of active movies and the sleep time = 1/cph.

5.2.2 Data Model

1. MOVIE

The MOVIE table stores the movie information.

Field	Type	Null	Key	Default	Extra
MOVIE_ID	INT(4)	NO	YES		AUTO INCREMENT
MOVIE_NAME	VARCHAR(50)	NO			
DELETE_TAG	INT(1)	NO		0	

MOVIE_ID: movie id
MOVIE_NAME: movie name
DELETE_TAG: indicates whether it is currently being processed or not (0: processed, 1: otherwise)

2. AUTHOR

The AUTHOR table stores the tweet author information.

Field	Type	Null	Key	Default	Extra
AUTHOR_ID	INT(9)	NO	YES		
AUTHOR_NAME	VARCHAR(20)	NO			
CHECK_TAG	INT(1)	NO		0	

AUTHOR_ID: id of the author who publishes the tweet
AUTHOR_NAME: name of the author who publishes the tweet
CHECK_TAG: indicates whether his or her friends and followers have already been found via Twitter REST API (0: no, 1: yes)

3. TWEET

The TWEET table stores the tweet information.

Field	Type	Null	Key	Default	Extra
TEXT_ID	BIGINT(10)	NO	YES		

AUTHOR_ID	INT(9)	NO			
MOVIE_ID	INT(4)	NO			
TEXT	VARCHAR(250)	YES			
POLARITY	VARCHAR(10)	YES		NA	

TEXT_ID: tweet status id
AUTHOR_ID: id of the author who publishes the tweet
MOVIE_ID: movie id
TEXT: contents of the tweet
POLARITY: sentimental polarity of a tweet (pos: positive, neg: negative, unknown: unknown. it is initially set to NA to indicate not processed)

4. THREAD_RESOURCE

The THREAD_RESOURCE table stores the available times to call the Twitter search method for the tweet-collection threads.

Field	Type	Null	Key	Default	Extra
THREAD_ID	INT(4)	NO	YES		
MOVIE_ID	INT(4)	NO			
CPH	NUMERIC(7,3)	NO			

THREAD_ID: thread id
MOVIE_ID: movie id
CPH: amount of the available calls per hour for this thread

5. THREAD_CALL

The THREAD_CALL table stores the time when tweet-collection threads call the Twitter search method.

Field	Type	Null	Key	Default	Extra
THREAD_ID	INT(4)	NO			
TIME	TIMESTAMP	NO		NOW()	

THREAD_ID: thread id
TIME: the time when some thread calls the Twitter API search method

5.2.3 Variables and Constants

1. Variables

Name	Default Value	Remark
search_limit	300	It is defined in the property file “config.properties”. See Chapter 5 “Property File” of the MovieOracle

		User Tutorial document for more details.

2. Constants

Name	Default Value	Remark
LANGUAGE_CODE	en	It represents “English”, which directs the Twitter API to only retrieve English tweets. More language codes can be found in the Twitter4J document [5].
TWEETS_PER_PAGE	100	It indicates the maximum number of tweets that the Twitter4J method (see Section 3.1) retrieves within one query.

5.2.4 Algorithm

1. The program gets all of the movie names from the MOVIE table with DELETE_TAG = 0.
2. It deletes all records in the THREAD_RESOURCE table.
3. For each movie acquired in 1, the program inserts the thread id (program generated), movie id, cph (search_limit /the total number of movies) to the THREAD_RESOURCE table and then starts a thread. Within each thread:
   (a) It sets the movie name as a query key, TWEETS_PER_PAGE as the number of returned tweets, and LANGUAGE_CODE as the tweet language code for the Query instance.
   (b) Until tweet collection on this movie ended (if DELETE_TAG of the movie equals to 1), repeat step i – v.
      i. In the THREAD_CALL table, in a synchronized call, check the total number of calls using the search method of Twitter4J within the past 1 hour. If this number is greater than search_limit, it sleeps for 1 second and then executes sub step i again. Otherwise, it executes ii.
      ii. Insert the thread id to the THREAD_CALL table (to record one call at the certain time).
      iii. Retrieve the tweets by the search method (see Section 3.1).
      iv. For each tweet found in iii, the program inserts the tweet text id, tweet author id, movie id, and tweet text to the TWEET table and inserts the tweet author id and author name to the AUTHOR table.
      v. It gets the current cph from the THREAD_RESOURCE table. The thread sleeps for 1/cph hour(s). After waking up, it executes i – v again.

5.3 Polarity Classifier

The implementation of the Polarity Classifier is supported by the two tables in 5.3.2 Data Model. They are TWEET and MOVIE. The specification of this part is in 3.3 of the Software Requirements Specification document.

5.3.1 Target and Requirements

The Polarity Classifier component classifies tweets' sentimental polarities and writes the results to the database. Since the tweet collection threads continually store the newest tweets, this component is implemented by a thread and periodically checks the database to find unclassified tweets. A tool called CI-Bayes (see Section 4) performs the basic classification job for each tweet.

5.3.2 Data Model

TWEET
(Check the TWEET table in 5.2.2 Data Model)

MOVIE
(Check the MOVIE table in 5.2.2 Data Model)

5.3.3 Variables and Constants

1. Variables

Name	Default Value	Remark
polarity_training_set	ClassifierTrainingSet	It is defined in the property file "config.properties". See Chapter 5 "Property File" of the MovieOracle User Tutorial document for more details.
check_tweet_interval	5	It is defined in the property file "config.properties" and indicates the time interval for classifying tweets. See Chapter 5 "Property File" of the MovieOracle User Tutorial document for more details.

2. Constants

Name	Default Value	Remark
REPLACE_NAME	movie	It indicates the constant string used to replace the real movie name in tweets. The replacement is for avoiding the possible positive or negative words in the movie name that may influence the polarity classification.

5.3.4 Algorithm

1. If the training data set (polarity_training_set folder) does not exist, the program prints a fatal error in the log file and quits. Otherwise, it uses the files in three subfolders of polarity_training_set to train the fisher classifier (see Section 4.1). The names of the three subfolders are pos, neg, and unknown.
2. It retrieves all of the tweet texts and corresponding movie names from the TWEET table joined with the MOVIE table where POLARITY = 'NA'. For each of the tweet texts and its corresponding movie name:
   (a) The program replaces the movie name in the text with REPLACE_NAME. Then it calls the method "getFisherProbability(Object tweet, String category)" of the fisher classifier to calculate the positive probability P, negative probability N, and unknown probability U of the tweet text, respectively.
   (b) The program uses the four rules in Section 4.2 to classify this tweet.
   (c) It updates the classified result (polarity) to the TWEET table.
3. It sleeps for check_tweet_interval minutes. After waking up, the program executes 2. again.

5.4 Potential User Finder

The implementation of the Potential User Finder component is supported by four tables in Section 5.4.2 Data Model. They are TWEET, AUTHOR, RELATED_TO_AUHOR, and POTENTIAL_USER. The specification of this part is in 3.4 of the Software Requirements Specification document.

5.4.1 Target and Requirements

The Potential User Finder component targets finding predictable users and calculating the percentages of three kinds of their related tweets. It continually checks the database to get new tweet authors. Since the rate of finding friends or followers is limited by the Twitter REST API, this component is implemented by a thread and needs to wait once exceeding the limitation.

5.4.2 Data Model

TWEET
(Check the TWEET table in 5.2.2 Data Model)

AUTHOR
(Check the AUTHOR table in 5.2.2 Data Model)

6. RELATED_TO_AUHOR
The RELATED_TO_AUHOR table stores the tweet authors (ids) and their corresponding friends and followers (ids).

Field	Type	Null	Key	Default	Extra
AUTHOR_ID	INT(9)	NO	YES		
USER_ID	INT(9)	YES	YES		
FLAG	INT(1)	YES	YES		

AUTHOR_ID: id of the author who publishes the tweet
USER_ID: twitter user id
FLAG: indicates whether a twitter user is a friend or follower of the tweet author (0: friend, 1: follower)

7. POTENTIAL_USER
The POTENTIAL_USER table stores the potential user information.

Field	Type	Null	Key	Default	Extra
MOVIE_ID	INT(4)	NO	YES		
USER_ID	INT(9)	NO	YES		
RELATIVE_POS_NUMBER	NUMERIC(6,5)	YES	NO		
RELATIVE_NEG_NUMBER	NUMERIC(6,5)	YES	NO		
RELATIVE_UNKNOWN_NUMBER	NUMERIC(6,5)	YES	NO		
POLARITY	VARCHAR(10)	YES	NO		

MOVIE_ID: movie id
USER_ID: twitter user id
RELATIVE_POS_NUMBER: relative number of positive related tweets
RELATIVE_NEG_NUMBER: relative number of negative related tweets
RELATIVE_UNKNOWN_NUMBER: relative number of unknown related tweets
POLARITY: sentiment prediction of the user on the movie (pos: positive, neg: negative, unknown: unknown)

5.4.3 Variables and Constants

1. Variables

Name	Default Value	Remark
related_tweets_threshold	6	It is defined in the property file “config.properties”. According to experiments, the accuracy of the decision tree with related_tweets_threshold < 6 is lower than related_tweets_threshold >= 6, but the there is no big difference between the accuracies of the decision tree with related_tweets_threshold = 6 and the decision tree with related_tweets_threshold > 6.
remain_hit	NULL	This value is from “twitter. getRateLimitStatus().getRemainingHits()”. It indicates the remaining number of Twitter REST API requests available. The getRemainingHits() method itself does not consume the available requests of Twitter REST API (see Section 3.1).

2. Constants

Name	Default Value	Remark
FRIEND_TAG	0	In the RELATED_TO_AUHOR table, it indicates the user (id) is the friend of the author (id).
FOLLOWER_TAG	1	In the RELATED_TO_AUHOR table, it indicates the user (id) is the follower of the author (id).
FRIEND_TAG & FOLLOWER_TAG		Currently, no distinction is made between friend and follower, but these two tags are included for future improvement.

5.4.4 Algorithm

1. The program gets the movie ids from the AUTHOR table join the TWEET table with CHECK_TAG = 0.
2. If the number of movie ids found in step 1 equals 0, sleep for 1 minute and executes 1 again. Otherwise, set rest_call to remain_hit/2 (for each author, the

program needs to call Twitter API method twice: “getFriendsIDs” method and “getFollowersIDs” method). For each movie id found in 1:

   (a) If rest_call equals to 0, the program executes step 3. Otherwise, it executes *(b)*.
   (b) Query the authors from the AUTHOR table join the TWEET table with CHECK_TAG = 0 and MOVIE_ID = movie id.
   (c) If the number of author ids is less than rest_call, keep all of the author ids. Otherwise, keep rest_call author ids. For each of the author ids:
      i. The program gets all of his or her friend ids via the method “twitter.getFriendsIDs(String screenName)” and inserts these ids, the author id, and FRIEND_TAG to the RELATED_TO_AUHOR table.
      ii. The program gets all of his or her follower ids via the method “twitter.getFollowersIDs(String screenName)” and inserts these ids, the author id, and FOLLOWER_TAG to the RELATED_TO_AUHOR table.
      iii. The program updates CHECK _TAG to 1 in the AUTHOR table where AUTHOR_ID = the author id in *(c)*.
   (d) Put the movie id used in *(b)* in a movie list and set rest_call to rest_call minus the number of authors kept in *(c)*.

3. If the movie list (which is set in 1*(d)*) is not null, then for each movie id in this list:
   (a) Retrieve the user ids from the RELATED_TO_AUHOR table join the TWEET table where MOVIE_ID = the movie id and group by USER_ID having the number of its corresponding AUTHOR_ID be greater than or equal to related_tweets_threshold.
   (b) The program clears the POTENTIAL_USER table with MOVIE_ID = the movie id. Then for each of the user ids in 3*(a)*:
      i. Retrieve the corresponding authors’ opinions (sentimental polarities) on this movie id from the RELATED_TO_AUHOR table join the TWEET table where USER_ID = the user id and MOVIE_ID = the movie id.
      ii. Calculate the percentage of positive, percentage of negative, and percentage of unknown related tweets for this user.
      iii. Insert the user id, the movie id, and these percentages to the POTENTIAL_USER table.
4. The program clears the movie list set in 1*(d)*. It invokes the decision tree to predict users whose POLARIY is null in the POTENTIAL_USER table.
5. If remain_hit equals to 0, sleep for 1 minute and execute step 5 again. Otherwise, execute step 1 - 5 again.

5.5 Decision Tree

The implementation of the Decision Tree component is supported by two tables introduced in 5.5.2 Data Model. They are POTENTIAL_USER and

DECISION_TREE_TRAINING. The specification of this part is introduced in 3.5 of the Software Requirements Specification document.

5.5.1 Target and Requirements

The Decision Tree component targets predicting potential users' opinions on movies and writing prediction results to the database. The decision tree is invoked by the Potential User Finder component, however the tree construction (training) starts just after the entire system starts (more explanation for training is in Section 2.2.1). When the tree is invoked, it also prints all users who hold positive attitudes to an indicated file.

5.5.2 Data Model

POTENTIAL_USER
(Check the POTENTIAL_USER table in 5.4.2 Data Model)

8. DECISION_TREE_TRAINING

The DECISION_TREE_TRAINING table stores the training data for the decision tree.

Field	Type	Null	Key	Default	Extra
TWEET_ID	BIGINT(10)	NO	YES		
RELATIVE_POS_NUMBER	NUMERIC(6,5)	YES	NO		
RELATIVE_NEG_NUMBER	NUMERIC(6,5)	YES	NO		
RELATIVE_UNKNOWN_NUMBER	NUMERIC(6,5)	YES	NO		
POLARITY	VARCHAR(10)	YES	NO		

TWEET_ID: tweet status id
RELATIVE_POS_NUMBER: relative number of positive related tweets
RELATIVE_NEG_NUMBER: relative number of negative related tweets
RELATIVE_UNKNOWN_NUMBER: relative number of unknown related tweets
POLARITY: sentimental polarity of a tweet (pos: positive, neg: negative, unknown: unknown)

5.5.3 Variables and Constants

1. Variables
   None

2. Constants

Name	Default Value	Remark

RECOMMENDATION	Recommendations.txt	It indicates the name of the file which contains twitter use ids with the specific movies they like.

5.5.4 Algorithm

The algorithm of the decision tree component can be divided into two parts: tree construction and tree invocation.

Construction

1. The program gets all of the records from the DECISION_TREE_TRAINING table. According to the ID3 algorithm [6], it uses these training data to construct a decision tree. The detailed implementation of the algorithm is described in Section 6.2.1

Invocation

1. Once invoked, the program retrieves the records from the POTENTIAL_USER table where POLARITY is null and then traverses the tree to predict potential users' sentiments on movies.
2. It updates the predicted polarities to the POTENTIAL_USER table.
3. The program then retrieves users who have positive opinions in the POTENTIAL_USER table and writes them to the indicated file RECOMMENDATION.

5.6 Command-line Interface

The command line interface is a part of the System Service component. Its implementation is supported by four tables introduced in 5.6.2 Data Model. They are POTENTIAL_USER, TWEET, THREAD_RESOURCE, and MOVIE. The specification of this part is in 3.1 of the Software Requirements Specification document.

5.6.1 Target and Requirements

The Command-line Interface targets providing commands to use the entire system.

5.6.2 Data Model

POTENTIAL_USER

(Check the POTENTIAL_USER table in 5.4.2 Data Model)

TWEET
(Check the TWEET table in 5.2.2 Data Model)

THREAD_RESOURCE
(Check the THREAD_RESOURCE table in 5.2.2 Data Model)

MOVIE
(Check the MOVIE table in 5.2.2 Data Model)

5.6.3 Variables and Constants

1. Variables

Name	Default Value	Remark
search_limit	300	It is defined in the property file “config.properties”. See Chapter 5 “Property File” of the MovieOracle User Tutorial document for more details.

2. Constants
NULL

5.6.4 Algorithm

1. It prints a prompt denotation and waits for screen input.
   (a) If the prefix of the input equals “predict”:
      i. It validates whether this prefix is followed by three file names which are separated by spaces.
      ii. Further validate that file 1 and file 2 exist and file 3 does not exist
      iii. If the validation fails, print the error and execute 1 again.
      iv. Otherwise, it gets twitter user ids from file 1 and gets movie names from file 2. For each movie and each user id, it executes the procedure A and B, and then prints the users’ sentiment prediction on the movies to file 3.
         A. The program returns the user’s opinion (polarity) from the POTENTIAL_USER table where MOVIE_ID = the movie id and USER_ID = the user id.
         B. If the user’s opinion does not exist in this table, it gets the major polarity from the TWEET table with MOVIE_ID = the movie id and calculates the simplistic prediction score (which is the maximum integer less than or equal to 10 * P, let P be the

highest percentage of sentimental polarities of such movie. Therefore the prediction score is between 3 and 10).

(b) If the prefix of the input equals “oracle”:
   - i. It validates whether this prefix is followed by a user id and a movie name which are separated by a space.
   - ii. Further validate that the user id is an integer.
   - iii. If the validation fails, print the error and execute 1 again.
   - iv. Otherwise, it executes procedures *(a)*iv A and *(a)*iv B*,* and then prints the prediction result to screen.

(c) If the prefix of the input equals “collect”:
   - i. It validates whether this prefix is followed by a file name.
   - ii. Further validate that the file does exist.
   - iii. If the validation fails, print the error and execute 1 again.
   - iv. Otherwise, it gets all movie names from the file.
   - v. The program then calculates the cph = search_limit /total number of movies (without duplicate movies) and updates all cph in the THREAD_RESOURCE table.
   - vi. For each movie found in iv, it checks whether it is in the MOVIE table where DELETE_TAG = 0 and MOVIE_NAME = the movie name. If yes, it prints a warning message. Otherwise, it executes A, B and C.
      - A. The program stores the new movie in the MOVIE table or updates DELETE_TAG to 0 where MOVIE_NAME = the movie name.
      - B. Put the cph and the movie (id) to the THREAD_RESOURCE table.
      - C. Start a thread (see the thread design in Section 5.2.4) to collect tweets on the new movie.

(d) If the prefix of the input equals “start”:
   - i. It validates whether this prefix is followed by a movie name.
   - ii. If the validation fails, print the error and execute 1 again.
   - iii. Otherwise, check whether the movie is in TWEET table with DELETE_TAG = 0 and MOVIE_NAME = the movie name.
   - iv. If yes, it prints a warning message and executes 1 again.
   - v. Otherwise, the program calculates the cph = search_limit/total number of movies and executes procedures *(c)*vi A, *(c)*vi B, and *(c)*vi C.

(e) If the prefix of the input equals “terminate”:
   - i. It validates whether this prefix is followed by a file name.
   - ii. Further validate that the file does exist.
   - iii. If the validation fails, it prints the error and executes 1 again.
   - iv. Otherwise, it gets all movie names from the file.
   - v. The program then calculates the cph = search_limit /total number of movies (without removed movies) and updates all cph in the THREAD_RESOURCE table.

        vi. For each movie, it checks whether it is in the MOVIE table with DELETE_TAG = 0 and MOVIE_NAME = the movie name. If no, it prints a warning message. Otherwise, it executes A and B.
            A. The program updates DELETE_TAG to 1 in the MOVIE table with MOVIE_NAME = the movie name.
            B. Delete the record in the THREAD_RESOURCE table with MOVIE_ID = the movie id.

(f) If the prefix of the input equals “stop”:
   i. It validates whether this prefix is followed by a movie name.
   ii. If the validation fails, print the error and execute 1 again.
   iii. Otherwise, it checks whether the movie is in the MOVIE table with DELETE_TAG = 0 and MOVIE_NAME = the movie name.
   iv. If no, print a warning message and execute 1 again.
   v. Otherwise, the program calculates the cph = search_limit /total number of movies (without the removed movie) and updates all cph in the THREAD_RESOURCE table. Then it executes procedures *(e)*vi A and *(e)*vi B.

(g) If the input equals “show movies”:
   i. The program gets all the movie names from the MOVIE table with DELETE_TAG = 0 and then prints them.

(h) If the input equals “show commands”:
   i. The program prints all the commands and their explanations (see Chapter 4 “Commands” of the MovieOracle User Tutorial document).

(i) If the input equals “exit”:
   i. The program stops and exits.

(j) The program executes step 1 again.

5.7 System Service

The implementation of the System Service component is not directly supported by any table. The specification of this part is introduced in 3.1 of the Software Requirements Specification document.

5.7.1 Target and Requirements

The System Service component invokes other components. It also provides parameter validation and log service for the entire project. This component is started once the system is started.

5.7.2 Variables and Constants

1. Variables

Name	Default Value	Remark
clear_interval	12	It is defined in the property file “config.properties”. See Chapter 5 “Property File” of the MovieOracle User Tutorial document for more details.

2. Constants
   None

5.7.3 Data Model

NULL

5.7.4 Algorithm

1. The program validates related_tweets_threshold, search_limit, clear_interval and check_tweet_interval in the property file “config.properties”. (In Chapter 5 “Property File” of the MovieOracle User Tutorial document, it discusses the selection of these variable values.) The validation includes:
   (1) related_tweets_threshold, an integer greater than 3.
   (2) search_limit, an integer no greater than 300.
   (3) clear_interval, a float number greater than 0.
   (4) check_tweet_interval, a float number greater than 0.
   If the validation passes, it executes 2. Otherwise, the program indicates the error and exits.
2. It prompts to input the database user name and password. If they are incorrect or the database service is not started, the program indicates the error and requires input again.
3. The program starts the Tweet Collection (Section 5.2) service and the Polarity Classifier (Section 5.3) Service. Then it constructs the Decision Tree (Section 5.5) and starts the Potential User Finder (Section 5.4) service.
4. For every clear_interval hours, a thread clears all of the records in the THREAD_CALL table an hour ago.
5. The program starts the Command-line (Section 5.6) service.

6. Details of Supplemental System

This part discusses two important theories which support the entire system. They are the sentiment analysis method and the decision tree method.

6.1 Sentiment Analysis [7]

Sentiment Analysis, also called Opinion Mining, is a cross research field of text mining, and machine learning [8]. It aims to find the text authors' attitudes on some topic. This project requires automatically determining the polarities of movie-review tweets from followers and friends. The ideal situation is finding an available twitter tool that can do this work. Otherwise, data-mining tools and methods will be considered.

6.1.1 Tweet Polarity

The movie reviews on twitter can be classified into positive, negative, and neutral [2]. Take reviews on the movie "Avatar" for example. "I saw Avatar. Very fun, great movie." is a positive tweet, while "I think I'm the only one on the planet that didn't like Avatar... Seemed a bit too much like an American history lesson + CG" is a negative tweet. However, there are still some ambiguous tweets which are difficult to classify as positive or negative. In this situation, they are considered neutral. For instance, "Yes, today I went to see the movie Avatar; in 3D. It was very well done, but not extraordinary...I give it a B- at best. Storyline is weak." is an ambiguous tweet, which might be considered positive or negative. In addition, when tweet authors do not display their opinions, their tweets on some movie can also be considered as neutral. For example, "Home from Aliante. Watched Avatar. Late." is neutral. Because of ambiguous tweets, even human beings have difficulties determining the sentimental polarity.

6.1.2 CI-Bayes

According to experimental comparisons of many programs, CI-Bayes is chosen for this project. It can be used by adding the jar file to the project classpath. CI-Bayes provides two major classification systems: Bayesian classifier and Fisher classifier.

---

[2] When talking about the tweet polarity, neutral and unknown are interchangeable terms in this document.

6.1.3 Bayesian Classification [9]

Bayesian classification is based on a statistical method called Bayes theorem. Assuming X describes a group of values on n attributes and H is a certain hypothesis. Then the fundamental formula of Bayes theorem can be represented as

P(H|X) = $\frac{P(X \mid H)P(H)}{P(X)}$. In this formula, P(H|X) is the posterior probability, which indicates the probability of hypothesis H given that X happens. P(X|H) is the posterior probability, which indicates the probability of X happening given that hypothesis H exists. P(H) is the prior probability, which indicates the probability of hypothesis H existing regardless of any X. P(X) is the prior probability, which indicates the probability of X happening regardless of any H.

Since there are multiple classes in a data set, the naive Bayesian classifier will choose the class having the highest posterior probability as the final result. Let the multiple classes be C_1, C_2, ..., C_m. According to the formula P(H|X) = $\frac{P(X \mid H)P(H)}{P(X)}$, an item with the attribute vector X belongs to the class C_i with the maximum P(C_i|X), which is equal to $\frac{P(X \mid C_i)P(C_i)}{P(X)}$. Only P($X$|$C_i$)P($C_i$) need to be considered in that P(X) indicates the probability of X happening regardless of any C_i and thus is a fixed value. Let the training data set be D, then P(C_i) = $|C_{i,D}|/|D|$, where $|C_{i,D}|$ is the number of items belonging to C_i in D. As to P(X|C_i), all attributes represented by X can be assumed as independent of one another. Let the attribute vector X be (x_1, x_2, ..., x_n). Thus, P(X|C_i) = $\prod_{k=1}^{n} P(x_k \mid C_i)$. P($x_k$|$C_i$) represents the probability of the value x_k on some attribute given all items of class C_i.

When using Bayesian classification on tweet sentimental analysis, each attribute refers to one particular word in a tweet and the hypothesis means the polarity of a tweet: positive, negative, or neutral. Suppose there is a tweet “Avatar is awesome”. The word “Avatar” appears in 100 percent of the positive training set, the word “is” appears in 30 percent of the positive training set, and the word “awesome” appears in 20 percent of the positive training set. Then the independent probability of them appearing in a positive tweet is 1×0.3×0.2 = 0.06. At the same time, the percentage of the three words appearing in the negative training set are 100 percent, 25 percent, and 0 percent, respectively, and their percentage in the unknown training set are 100 percent, 35 percent, and 1 percent. Therefore the independent probability of them appearing in a negative tweet is 1×0.25×0 = 0 and in an unknown tweet is

1×0.35×0.01 = 0.0035. Further assume the weights of positive, negative, and unknown tweets on training set are 0.4, 0.2, and 0.3, respectively. So the probability of a positive tweet is 0.06×0.4 = 0.024, the probability of a negative tweet is 0×0.2 = 0, and the probability of an unknown tweet is 0.0035 × 0.3 = 0.00105. Since this tweet has the highest probability of being positive, it is considered a positive review on Avatar.

6.1.4 Fisher Classification [10]

An alternative classification system considered in CI-Bayes is Fisher classification. This classification is based on Fisher's method. Assuming k is the number of attributes and the attribute vector X is (x_1, x_2, ..., x_k). Then the fundamental formula of Fisher's method can be represented as $X^2 = -2\sum_{i=1}^{k}\log_e(p_i)$. In this formula, p_i is the probability of hypothesis H given that x_i happens. The formula can also be written as $X^2 = -2\log_e\prod_{n=1}^{k}p_i$.

The Fisher classifier is different from naive Bayesian classification in that it estimates the probability of a hypothesis (an item belong to some class) for each attribute on an item. Let multiple classes be C_1, C_2, ..., C_m and $j \in [1, m]$. Given an item, p_i can be calculated as the ratio of the frequency of x_i in the data with class C_j to the frequency of x_i in the data of all classes. For m attributes, we can get m*p_i values. With the formula $X^2 = -2\sum_{i=1}^{k}\log_e(p_i)$, the probability of this item belonging to C_j is calculated. Similar to Bayesian classification, an item with the attribute vector X belongs to the class C_j with the maximum probability.

Also, when using Fisher classification on tweet sentimental analysis, each attribute refers to one particular word in a tweet and the hypothesis means the polarity of a tweet: positive, negative, or neutral. Suppose there is a tweet "Avatar is awesome". The word "Avatar" appears in 100 percent of the positive training set, 100 percent of the negative training set, and 100 percent of the unknown training set. The word "is" appears in 30 percent of the positive training set, 25 percent of the negative training set, and 35 percent of the unknown training set. The word "awesome" appears in 20 percent of the positive training set, 0 percent of the negative training set, and 1 percent of the unknown training set. Then the probability of a positive tweet is $\frac{1}{(1+1+1)} \times \frac{0.3}{(0.3+0.25+0.35)} \times \frac{0.2}{(0.2+0+0.01)}$ = 0.106, the probability of a negative tweet is $\frac{1}{(1+1+1)} \times \frac{0.25}{(0.3+0.25+0.35)} \times \frac{0}{(0.2+0+0.01)}$ = 0, and the probability of a unknown tweet is

$\frac{1}{(1+1+1)} \times \frac{0.35}{(0.3+0.25+0.35)} \times \frac{0.01}{(0.2+0+0.01)}$ = 0.006. This tweet is considered a positive review on Avatar.

6.1.5 Type I and type II errors [11]

Type I and type II errors are used to test the correct rate of one classifier. Type I error, also known as “false positive”, occurs when some hypothesis is accepted but actually it is not true. For example, one tweet is classified as positive but in fact it is negative or unknown. In this situation, accepting the hypothesis “this tweet is positive” makes a type I error. Type II error, also known as “false negative”, occurs when some hypothesis is rejected but actually it is true. For example, one tweet is classified as negative or unknown but in fact it is positive. In this situation, rejecting the hypothesis “this tweet is positive” makes a type II error. Type I error rate is the proportion of the events which actually should reject a hypothesis but accept it. For example, let the total number of both negative and unknown tweets be 40. But one classifier mistakenly determines that 10 of them are positive, then the Type I error rate for determining positive tweets is 10/40 = 0.25. Type II error rate is the proportion of the events which actually should accept a hypothesis but reject it. For example, let the total number of both positive tweets be 20. But one classifier mistakenly determines that 5 of them are not positive, then the Type II error rate for determining positive tweets is 5/20 = 0.25.

6.1.6 Comparison of naive Bayesian and Fisher classification

A Java project CI-Bayes provides two major classification features: one is a naive Bayesian classifier and one is a Fisher classifier. Five-fold cross validation is used to determine the correct rate of the two classifiers. The error rates for naive Bayesian classifier are 0.30666667, 0.34, 0.38, 0.3, 0.34, for Fisher classifier are 0.23333333, 0.26, 0.22, 0.26666668, 0.22666667. Tables 3 and 4 describe Polarity Distribution of two classifiers.

The training is chosen to represent real tweets, by setting the proportion of positive, negative, unknown, and neutral tweets close to 4:2:3. The data includes 370 positive tweets, 180 negative tweets, and 240 unknown tweets (including neutral tweets).

Fold	Predicted Neg/Pos/Unk of Neg	Predicted Neg/Pos/Unk of Pos	Predicted Neg/Pos/Unk of Unk
1	0.5/0.3611/0.1388	0.04/0.8533/0.1066	0.1041/0.2916/0.6041
2	0.5833/0.2777/0.1388	0.0133/0.8933/0.0933	0.1041/0.5416/0.3541
3	0.5277/0.25/0.2221	0.0266/0.84/0.1333	0.0625/0.5208/0.4166

4	0.5555/0.2777/0.1666	0.0266/0.8666/0.1066	0.0625/0.5208/0.4166
5	0.5/0.3333/0.1666	0.0266/0.8/0.1733	0.1041/0.4791/0.4166

Table 3: Polarity Distribution of naive Bayesian Classifier Prediction

Fold	Predicted Neg/Pos/Unk of Neg	Predicted Neg/Pos/Unk of Pos	Predicted Neg/Pos/Unk of Unk
1	0.6944/0.1111/0.1944	0.1333/0.72/0.1466	0.2083/0.0625/0.7291
2	0.8889/0.0833/0.0277	0.0667/0.72/0.2133	0.1458/0.2291/0.6249
3	0.9166/0.0277/0.0555	0.0533/0.7466/0.1999	0.1041/0.2708/0.6249
4	0.75/0.0555/0.1943	0.08/0.76/0.1599	0.1875/0.3125/0.4999
5	0.75/0.0555/0.1943	0.0666/0.72/0.2133	0.2083/0.0833/0.7082

Table 4: Polarity Distribution of Fisher Classifier Prediction

6.1.7 Conclusion

From Table 3, the positive type I error rate of Fisher's classifier is much lower than naive Bayesian classifier, although its negative type I error rate is slightly higher than naive Bayesian classifier. As to type II error, table 4 displays that the performance of Fisher's classifier is much better on determining negative, while its positive error rate is close to naive Bayesian classifier. Since negative reviews in real tweets are few on most movies, correctly determining the negative tweets become very important. Furthermore, the total error rate of Fisher's classifier is lower than naive Bayesian classifier in each fold of five-fold cross validation. Therefore, the Fisher classifier is selected as the method to automatically determine the polarities of tweets.

6.2 Decision Tree [12]

A decision tree has two kinds of nodes: internal nodes and leaf nodes. Every node points to a set of tuples. In addition, each internal node represents a test on one attribute and its branches are created according to different values or ranges of values of the test result. For example, one internal node represents the test on an attribute "percentage of its negative related tweets" and its result has two values, one for all tuples with percentage of negative related tweets "$\leq$ 0.45" and one for all "> 0.45". Leaf nodes are labeled with a classification, also called the target attribute value. For example, leaf nodes are labeled by a sentimental polarity value so that each has one of three possible values "negative", "positive" and "neutral". The decision tree used in this project is restricted to a binary tree. However it could be a n-way splitting (n > 2) tree in other situations. The figure 6.1 depicts a sample decision tree.

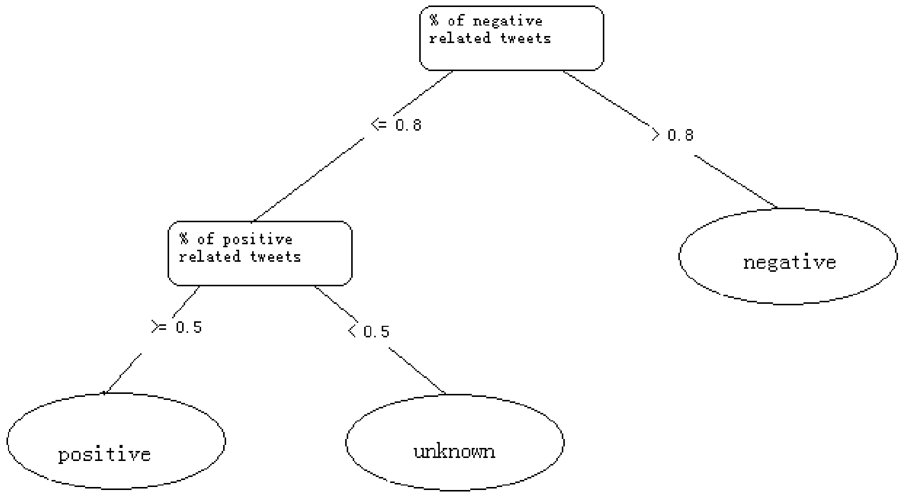

Figure 6.1

A decision tree builds a tree model on training data in order to predict unknown data. Training data in this project come from the super tweet set[2] whose sentimental polarity are already known[3], while test data comes from the super tweet set with unknown sentimental polarity. Each tuple in the training data set represents a tweet and it has four attributes: percentage of its negative related tweets, percentage of its positive related tweets, percentage of its unknown related tweets, and its own sentimental polarity. Each tuple of the test data set only has the former three attributes of the training set, but does not have the last one. Sentimental polarity is a target attribute that will be predicted for the test set.

The construction of a decision tree starts with one single node, pointing to all of the tuples in the training set. If all tuples on this node belong to the same class (have the same target attribute value), it becomes a leaf node and labeled with that class. Otherwise, the ID3[6] method (which will be described in detail in the next paragraph) is used to select a “best” split attribute and a “best” split point. On the split attribute, the data set is then partitioned into two subsets according to the split point (a threshold). Each partition becomes a child of the current node, and this procedure is repeated until all children are leaf nodes. In this project, because a tuple has very limited attributes (totally three), the construction does not remove an attribute after selecting it as a split attribute. In other words, a selected attribute will also be reconsidered in the next recursion.

In the ID3 method, given a data partition (a set of tuples) D, a classification (a target attribute value) C_i, and $C_{i,D}$ be the set of tuples classified as C_i in D. The total information needed to classify a tuple can be represented by

Info(D) = $-\sum_{i=1}^{m} p_i \log_2(p_i)$, where p_i is $|C_{i,D}|/|D|$ ($|C_{i,D}|$ and $|D|$ indicate the number of tuples in $C_{i,D}$ and D, respectively). ID3 selects a "best" attribute and a "best" split point (threshold) that generate two data sets D_1 and D_2 to minimize $Info_A(D)$ = $\sum_{j=1}^{n} \frac{|D_j|}{|D|} \times Info(D_j)$, which is the information needed to further classify the data after the partition.

If tuples of D have disjoint values on attribute A, to get the "best" split point, these values will be sorted in increasing order. Then consider the midpoint (mean value) between each pair of adjacent values. For every possible midpoint, the tuples are divided into two partitions: one for lower than or equal to it and one for higher than it. Therefore, if the tuples on attribute A have n distinct values, ID3 will test (n-1) partitions. Further, if all tuples have m attributes and each attribute has n_i values (for i = 1,...,m), to acquire the "best" attribute and the "best" split point, ID3 needs to consider $\sum_{i=1}^{m}(n_i - 1)$ possible partitions.

Finally, information that can be acquired after a partition is defined as Gain(A) = Info(D) - $Info_A$(D). Consequently, the algorithm selects the split attribute and the split point that maximize Gain(A), or minimize $Info_A$(D) since Info(D) is fixed.

6.2.1 Algorithm

In the following algorithm, the original input d is the training set. All tuples of this set are acquired from a database.

```
CONSTRUCT(d)
(1) create a new node containing all tuples of d
(2) if all tuples have the same class, then node = leaf and value = class of the classification, return
(3) otherwise, over all 3 attributes, use ID3 method to get the attribute and the split point with highest information gain
(4) if the highest information gain $\leq \varepsilon$, label this node as a leaf with value = class of the most common class in the set, return
(5) on the "best" split attribute, use the "best" split point p to divide the set into two partitions A and B: all tuples of A having values of split attribute $\leq$ p and all tuples of B having values of split attribute > p
(6) for each partition, recursively invoke the procedure CONSTRUCT(d)
```

References

[1] Twitter Inc. Engineers. REST API Resources | Twitter Developers https://dev.twitter.com/docs/api Twitter Inc.
[2] Yusuke Yamamoto. Twitter4J - A Java library for the Twitter API http://twitter4j.org Open Source
[3] Twitter Inc. Engineers. Authentication & Authorization | Twitter Developers https://dev.twitter.com/docs/auth Twitter Inc.
[4] Joseph Ottinger. CI-Bayes https://ci-bayes.dev.java.net Open Source
[5] Yusuke Yamamoto. Twitter4J - JavaDoc http://twitter4j.org/en/javadoc.html Open Source
[6] Jiawei Han and Micheline Kamber. Data Mining: Concepts and Techniques, Second Edition (Chapter 6.3.2)
[7] Yelena Mejova. Sentiment Analysis: An Overview. 2009
[8] Bing Liu. Opinion Mining. 2008
[9] Jiawei Han and Micheline Kamber. Data Mining: Concepts and Techniques, Second Edition (Chapter 6.4.1 and 6.4.2)
[10] Toby Segaran. Programming Collective Intelligence, First Edition (Chapter 6)
[11] Tom Rogers. Amazing Applications of Probability and Statistics
[12] Jiawei Han and Micheline Kamber. Data Mining: Concepts and Techniques, Second Edition (Chapter 6.3.1)