Detecting Wikipedia Vandalism

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

in

Information Technology

under the Supervision of

Ms Reema Aswani

By

Disha Sharma

Roll No- 111427

to

Jaypee University of Information and Technology

Certificate

This is to certify that project report entitled "**Detecting Wikipedia Vandalism**", submitted by **Disha Sharma** in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Supervisor's signature:

Supervisor's Name: **Ms Reema Aswani**

Designation: **Assistant Professor**

Date:

Acknowledgement

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. It not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

A special gratitude I give to our final year project supervisor , Ms. Reema Aswani , whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project, for the guidance and constant supervision as well as for providing necessary information regarding the project .

Furthermore, I have to appreciate the guidance given by other supervisor as well as the panels especially in my project presentation that has improved my presentation skills thanks to their comment and advices. My thanks and appreciations also to the people who have willingly helped me out with their abilities.

Name of the student : Disha Sharma

Date:

Table of Content

List of Tables

Abstract

Collaborative online social media (CSM) applications such as Wikipedia have not only revolutionized the World Wide Web, but they also have had a hugely positive effect on modern free societies. Unfortunately, Wikipedia has also become target to a wide-variety of vandalism attacks. Most existing vandalism detection techniques rely upon simple textual features such as existence of abusive language or spammy words. These techniques are ineffective against sophisticated vandal edits, which often do not contain the tell-tale markers associated with vandalism.

A plethora of methods have been developed within the Wikipedia and the scientific community to tackle this problem.This project has participated in this effort and proposes a content context-aware vandalism detection framework. The main idea is to quantify how well the words contained in the edit fit into the topic and the existing content of theWikipedia article.

.

Chapter 1

Introduction

1.1 What is Wikipedia

Wikipedia is an online encyclopedia that is free, collaborative, multilingual and global-scale. Free because anyone is free to use, copy, redistribute and modify Wikipedia content, even with commercial purposes, as long as the result is also shared with the same license. Collaborative because Wikipedia contents are created by the collaboration of thousands of individuals. Anyone can edit Wikipedia, even without being registered, and participate in the discussions about content and policies. Multilingual because there are editions of Wikipedia in 240 languages and growing. Global-scale because in its 10 years of life, Wikipedia has had an enormous growth. Today, it is the most popular source of encyclopaedic knowledge and one of the most visited websites on the Internet, with 365 million estimated readers. Only the English edition contains more than 3 million articles,over 13 million registered users and 130 thousand active users.

In short, the success of Wikipedia is also a key factor for the development of a wide range of academic, social and commercial projects beyond Wikipedia.

1.2 What is Vandalism

The fact that anyone can edit Wikipedia at any time with very little practical restrictions is at the core of its success and, at the same time, it is one of its main sources of trouble. By guaranteeing any person freedom to edit its contents, Wikipedia has become a target for pranksters and, with its increasing popularity, for spammers, lobbyists and other people interested in self-promotion, manipulation and propaganda. This has a wide-ranging negative impact in Wikipedia itself and all applications that use Wikipedia as a knowledge source. The phenomenon of vandalism can be defined as, any addition, removal, or change of content made in a deliberate attempt to compromise the integrity of Wikipedia. Common types of vandalism are the addition of obscenities or crude humour, page blanking, and the insertion of nonsense into articles. Any good-faith effort to improve the encyclopaedia, even if misguided or ill-considered, is not vandalism. Even harmful edits that are not explicitly made in bad faith are not vandalism.

1.3 Why does Vandalism matter

Considering the increasingly important role that Wikipedia is playing in the modern world, it is important to ensure the trustworthiness of the information that gets shared on it. Unfortunately, the very foundational

features of Wikipedia namely end-user anonymity and low information sharing barrier have made it susceptible to a variety of *vandalism attacks.*

Studies show that around 5% of Wikipedia edits involve vandalism. Some of these edits were not rectified for several hours (in some, albeit in frequent, cases even days). In addition to exposing false information to Wikipedia users, vandalism has the potential to inflict wider damage. It can cause progressive degradation of quality of information which can lead to frustration among honest contributors, some of whom may loose interest in contributing content and participating in Wikipedia activities.

More importantly, vandalism can create social tensions and may even lead to violence in certain regions of the world. Thus, it is important to develop effective techniques for detecting vandalism in Wikipedia as well as other CSM applications.

Chapter 2

Wikipedia Vandalism

2.1 Kinds of Vandalism

Vandalism is a highly subjective and wide concept. There have been attempts to give a concise definition by creating taxonomies of vandalism. There are many kinds of vandalism,as shown in Table 2.1.
Tightly attached to the concept of vandalism are good and bad faith, which are terms regularly used in the Wikipedia community. However, from a computational point of view, we are actually studying vandalism as damage to the encyclopedia, regardless of intentions and leaving
Judgemental issues to human experts.

Tightly attached to the concept of vandalism are good and bad faith, which are terms regularly used in the Wikipedia community. However, from a computational point of view, we are actually studying vandalism as damage to the encyclopedia, regardless of intentions and leaving judgamental issues to human experts.

Table 2.1: Summary of types of vandalism

Type	**Description**
Blanking	Removing all or significant parts of a page's content without any reason.
Edit summary vandalism	Making offensive edit summaries in an attempt to leave a mark that cannot be easily expunged from the record.
Hidden vandalism	Any form of vandalism not visible in the final article but visible during editing.

Image vandalism	Uploading shock images, inappropriately placing explicit images on pages, or simply using any image in a way that is disruptive.
Link vandalism	Adding or changing internal or external links on a page to disruptive, irrelevant, or inappropriate targets.
Illegitimate page creation	Creating new pages with the sole intent of malicious behaviour.
Page lengthening	Adding very large amounts of content to a page so as to make the page's load time abnormally long.
Page-move vandalism	Changing the names of pages to disruptive, irrelevant and inappropriate names.
Silly vandalism	Adding profanity, graffiti or patent nonsense to pages.
Sneaky vandalism	Vandalism that is harder to spot, or that otherwise circumvents detection, including adding plausible misinformation and hiding vandalism through multiple edits.
Spam external linking	Adding links to irrelevant sites after having been warned.
Template vandalism	Modifying the wiki language or text of a template in a harmful or disruptive manner.

2.2 Vandalism Statistics and Impact

The Wikipedia community conducts its own quantitative and qualitative studie on vandalism. Study 1 consisted of manually checking 100 random articles with a total of 668. Observed vandalism constituted a 4.6%. The observed time period comprised 2004, 2005 and 2006 and vandalism percentage appeared to be stable, oscillating between 3% and 6% of total edits. The most common vandalism type was obvious vandalism (83.87%) followed by deletion vandalism (9.68%). Currently, the most accurate estimation of vandalism in the English edition of Wikipedia is around 7% of all edits (Potthast 2010). If we consider that there were 10 million edits between August 20 and October 10 2010, which makes almost 200 thousand edits per day on average2, we can assume the order of magnitude of vandalism edits per day is 10^4.

According to Wikipedia's Study 1, 96.77% of all vandalism edits were performed by unregistered users. In 74.19% of cases, vandalism was reverted by a registered user.Another important statistic is how much time vandalism remains in Wikipedia and how many people view it. It is estimated that mass deletions remain 7.7 days on average with a median time of 2.8 minutes,while mass deletions involving obscenities remain 1.8 days on average with a median time of 1.7 minutes.

Another important statistic is how much time vandalism remains inWikipedia and how many people view it. Viégas, Wattenberg, and Dave (2004) estimated that mass deletions remain 7.7 days on average with a median time of 2.8 minutes, while mass deletions involving obscenities remain 1.8 days on average with a median time of 1.7 minutes. Priedhorsky et al. (2007) further studied the problem with results consistent with those by Viégas, Wattenberg, and Dave (2004), and estimated that the probability that a view

of Wikipedia between 2003 and 2006 included damaged content was of 0.0037. This probabily can be translated to 188 million views of vandalism during the studied perior.

2.3 Vandalism Impact: An Anecdote

To further illustrate the impact of vandalism, we expose an anecdote. On March 14th 2010, a prankster edited the Wikipedia article of a small belgian town:Kaster. After this edit, Kaster's article introduction was the following: Kaster is a village in Belgium, part of the municipality of Anzegem.Recently, the town made headlines when a serial rapist dressed as a Blastoise Pokemon raped and killed 65 men.

Most of the time, this would have been corrected quickly and nobody would have noticed. However, the edit was not reverted until April 9th 20105. During that month, Google's spider fetched the article and indexed it its database and as a result, typing define:kaster in Google's search engine would show up the above prank as the definition for Kaster.

What started as a small prank by two brothers ended up being an embarrassing thing forWikipedia and Google. Leaving aside the fun that this provided to many people, Wikipedia needs to put measures in place to fight vandalism and prevent damages on the credibility of the project.

Chapter 3

Wikipedia Vandalism Detection

3.1 Practical Tools Against Vandalism

3.1.1 Anti-vandalism Patrolling

The main force against vandalism is people who manually checks latest changes made to Wikipedia and review them to find vandalism and revert it. This activity is known as patrolling.

The classic method of patrolling is opening a browser tab with the list of recent changes and skim through the list, then open in other tabs suspicious edits, check them and revert them if necessary.

3.1.2 Patrolling Assistance

A wide variety of tools have been developed to assist patrollers in their work. These range from tools aimed at browsing and editing Wikipedia in a faster and more convenient way, such as Twinkle or Huggle, to automatic detection systems that work with human supervision, such as STiki.

3.1.3 Automatic Systems, Bots and Edit Filters

Automatic detection systems are designed to work with very limited human intervention or no intervention at all. In practice, there are two ways of implement them: as bots or edit filters.

On one hand, bots operate autonomously as agents external toWikipedia, and as such, they detect and revert vandalism some time after it is performed.

On the other hand, edit filters are a recent addition to the MediaWiki, deployed since 2009. They look for common patterns of vandalism at the edit time.If the edit matches one of these patterns, MediaWiki will reject it. The advantage of this approach is that when a vandalism edit is detected, it is rejected before it takes effect.

3.2 Why consider context?

One of the central limitations of traditional vandalism detection techniques is that most of them treat edits as independent and isolated pieces of texts. Because of this, most of them just focus on the text that appears in the edit. However, edits in Wikipedia are not isolated pieces of text. They occur in certain *context*, and hence the contextual attributes are an integral part of an edit's characteristics. For instance, an edit occurs on a certain version of an article. Thus, the edit cannot be completely characterized without including the content of the article at the time the edit occurred. In fact, the edit may become meaningless if it were to be performed on a different article or a different version of the same article.

In addition to article and version, an edit carries with it several other powerful contextual attributes. These include the identity (or lack thereof) of the user performing the edit, the previous history of edits performed by the user, the geographical location from where the edit originated, and the time at which the edit was performed. Many of these contextual attributes can be very powerful features in identifying vandalism. The importance of context is evident by the fact that even humans (implicitly) rely upon context when identifying vandalism. For example, most humans will immediately identify an edit containing the word "Nazi" as vandalism if the edit is on, say, President Obama's Wikipedia page, whereas they will not classify the same edit as vandalism if it is on Goebbels' page. The human is implicitly relying on whether the edit fits into the overall context of the article to determine whether it is vandalism.

There are many challenges to utilizing context for vandalism detection. First, we need to identify contextual attributes that have strong distinguishing capabilities. Second, context is often an abstract

concept, and for machines to understand and process it, context has to be made *quantifiable*. This means that we have to not only invent meaningful metrics for various contextual attributes, but also devise measurement mechanisms. Third, we need to design efficient and scalable vandalism detection techniques that utilize these quantifiable contextual attributes.

3.3 Problem Definition and Notation

A revision r is the state of an article in a given point of its history. We use r^- and r^+ to denote a past or future revision with respect to r, respectively.Using subindices r_i, r_{i+1} or r_{i_1} to denote specific past or future revisions. An edit e is the transition between two consecutive revisions. The Wikipedia vandalism detection task consists in decide whether a given edit e is vandalism or not. From the point of view of machine learning, given the set of E of all edits,

- A corpus $E_c \rightarrow E$ of labeled edits.
- An edit model α: $E \rightarrow E$ that maps each edit e onto a feature set e quantifying characteristics of e that are useful for discriminating between vandalism and non-vandalism edits.
- A classifier $c : E \rightarrow [0,1]$. The result of this classifier is the confidence of a given edit e being vandalism.
- A threshold T is defined so that any $c(e) \geq T$ indicates vandalism and $c(e) \leq T$ indicates otherwise, we check whether it is vandalism or not by
- For any unseen edit e that belongs to $E\ E_c$,check whether it is vandalism or not by computing $c(\alpha(e)) > T$.

3.3.1 Immediate and Historic Detection

Vandalism detection includes two different tasks: immediate8 and historic detection. Immediate detection is the most extended: detecting vandalism right after it happens. The historic variant is detecting vandalism at any point in the past.The technical difference between them is that, in the case of historic detection, information about everything that happened after the vandalism act is available to the system. Immediate detection is the most applied and useful to maintain Wikipedia clean of vandalism. The interest in historic detection is that much higher performance can be achieved, making it useful for building corpora to train immediate detection systems and also creating clean snapshots of Wikipedia, by selecting revisions of each articles that are guaranteed to be vandalism-free.

3.4 Corpora

For the best of our knowledge, there are sixWikipedia vandalism corpora. All our work used the PAN-WVC-10 corpus, although we will present all the six corpus for reference.

3.4.1 Webis-WVC-07

The Webis Wikipedia vandalism corpus11, or Webis-WVC-07, is the first public Wikipedia vandalism corpus reported in the literature. It consists of 940 edits annotated by humans, 301 of them annotated as vandalism. (Potthast and Gerling 2007; Potthast, Stein, and Gerling 2008).

3.4.2 Chin 2010

This corpus was built and used for (Chin et al. 2010). It was built based on the Wikipedia revision history up to February 24th, 2009 and it consists of the full history of two of the most vandalized pages12: Abraham Lincoln (8,816 revisions) and Microsoft (8,220 revisions).Annotation was performed in an active learning fashion. A first classification model was built using the Webis-WVC-07 corpus, and that model was used to get a rank of the top 50 candidates to be vandalism. An annotator revised these candidates and annotated them. The annotated edits were added to the training corpus and the process was repeated iteratively.
This annotation method makes (Chin et al. 2010) an interesting approach to solve the problem of annotating a corpus big enough to be used for supervised classification.

3.4.3 West 2010

West, Kannan, and Lee 2010 use a unique approach to annotate their corpus13. In Wikipedia, some privileged users have the right to revert an edit using a singleclick feature called rollback, used to undo blatantly unproductive edits.

The authors define an offending edit as one that was reverted using the rollback function.Although this is only a small portion of vandalism edits, this approach results in a very high confidence for positive annotations. The corpus contains 5,713,762 edits labeled as blatantly unproductive using the described automaitc method; it also contains 5,291 vandalism edits that were manually annotated. This makes West 2010 the largest Wikipedia vandalism corpus reported until now.

3.4.4 PAN-WVC-10

The PAN Wikipedia Vandalism Corpus 201014, or PAN-WVC-10, is the successor of Webis-WVC-07 (Potthast 2010). It consists of 32,439 edits, of which 2,394 are annotated as vandalism. Amazon Mechanical Turk was used to distribute the task amongst hundreds of human annotators. Each edit was annotated by 3 people. If they did not agree, the edit was annotated by 3 more people. This process was repeated until every edit had an annotation with more than 2/3 of inter-annotator agreement. After 8 iterations, there were 70 tie edits that were reviewed by the corpus authors.

Due to its use in the 1st International Competition on Wikipedia Vandalism Detection (Potthast, Stein, and Holfeld 2010) it is one of the most widely used corpus in the scientific literature.
In this project, I will use a modified version of PAN-WVC-10 where 157 edits were removed. This was because of these edits were deleted from the Wikipedia History at the time of writing (Adler et al. 2011). Statistics for our corpus version are: 32,282 total edits, with 2,395 vandalism edits.

3.4.5 PAN-WVC-11

The PAN Wikipedia Vandalism Corpus 2011, or PAN-WVC-11, is a supplement to PAN-WVC-10. It is the first multilingual corpus, including sections for English, German and Spanish.The English section consists of new 9985 annotated edits of the same time period as those compiled for PAN-WVC-10. 1144 of them are annotated as vandalism.The German section consists of 9990 edits, 589 of them annotated as vandalism. The Spanish section consists of 9974 edits, 1081 of them annotated as vandalism.

3.4.6 ClueBot-NG dataset

ClueBot-NG dataset is an ever evolving one. Through its online review interface a multitude of Wikipedia users annotate edits as vandalism, constructive or skipped. The final classification is decided as follows:

- A minimum of 2 annotators agreeing is required for the edit to be considered as annotated.
- If more than a half of annotators skipped the edit, it is annotated as skipped.
- If, at least, constructive annotations are thrice the vandalism annotations,the edit is annotated as constructive.
- If, at least, vandalism annotations are thrice the constructive annotations, the edit is annotated as vandalism.
- If none of the previous criteria is met, the edit is not considered as annotated and therefore it is not added to the final dataset.

  The strong point of this corpus is that it is annotated by experts. Therefore, we can expect a high quality in annotation and compliance with Wikipedia standards. This is an advantage over PAN-WVC-10, whose annotations might be less reliable; and over West 2010, which has a high amount of false negatives. However, its size is one order of magnitude below PAN-WVC-10 and four below West 2010.

3.4.7 Wikipedia dumps

Wikimedia offers XML and SQL dumps of the entire database for all its projects. These dumps include the full revision history of every article, along with other information. This is a resource commonly used to build Wikipedia vandalism corpora and detection systems.

3.4.8 Wikipedia User Contribution Dataset

Javanmardi, Lopes, and Baldi (2010) created a dataset24 of content insertions and deletions per user. This dataset comprehends all Wikipedia insertions and deletions since its creation to January 30th, 2010. This is a valuable approach for user reputation methods.

CHAPTER 4

Literature Review

4.1 Research Papers

Paper 1

Name	Wikipedia Vandalism Detection: Combining Natural Language, Metadata, and Reputation Features
Description	Integrated three approaches to detect vandalism. They are - spatio- temporal analysis of metadata (STiki), a reputation-based system (Wiki-Trust), and natural language processing features.The resulting joint system improved the state-of-the-art from all previous methods and establishes a new baseline for Wikipedia vandalism detection.
Authors	B. Thomas Adler, Luca de Alfaro, Santiago M. Mola-Velasco, Paolo Rosso, Andrew G. West
Approaches	Spatio-temporal analysis of metadata, Reputation-based system, Natural language processing features.
Algorithm	Random forest

Paper 2

Name	Automatic Wikipedia Vandalism Detection
Description	Discussed the characteristics of vandalism as humans recognize it and develop features to render vandalism detection as a machine learning task. Compiled a large number of vandalism edits in a corpus, which allowed for the comparison of existing and new detection approaches.
Authors	Martin Potthast, Benno Stein, and Robert Gerling
Approaches	Logitboost
Algorithm	Logistic regression

Paper 3

Name	Automatic Vandalism Detection in Wikipedia: Towards a Machine Learning Approach
Description	Investigated the possibility of using machine learning techniques to build an autonomous system capable to distinguish vandalism from legitimate edits. Highlighted the results of a small but important step in this direction by applying commonly known machine learning algorithms using a straightforward feature representation.
Authors	Koen Smets,Bart Goethals and Brigitte Verdonk
Approaches	Applied two machine learning algorithm
Algorithm	Naïve Baye's Classifier, Probablistic sequence modelling

Paper 4

Name	Detecting Wikipedia Vandalism via Spatio-Temporal Analysis of Revision Metadata
Description	Leveraged the spatio-temporal properties of revision metadata to detect vandalism. An administrative form of reversion called rollback was used which enablesd the tagging of malicious edits, which were contrasted with non-offending edits in numerous dimensions.
Authors	Andrew G West, Sampath Kannan and Insup Lee
Approaches	Spatio-temoral features
Algorithm	SVM

Paper 5

Name	A Content-Context-Centric Approach for Detecting Vandalism in Wikipedia
Description	Proposed a content context-aware vandalism detection framework. The main idea was to quantify how well the words contained in the edit fit into the topic and the existing content of theWikipedia article.
Authors	9th ieee international conference
Approaches	WWW co-occurrence probability,top-ranked co-occurrence probability
Algorithm	AdaBoost ,Naïve Bayes ,Decision Tree

4.2 Integrated Literature review

Various research have been proposed with the same aim i.e. creation of an antivandalism bot for the detection of Wikipedia. The basic idea was the same with tuning some features and introducing others , explored features based on word lists, expanding them beyond vulgarisms and improving results by creating new categories. Different well known machine learning classifiers were discussed ,varied the parameters to observe changes in performance. Mostly a 10 fold cross validation on the training data and measured precision (P), recall (R), F-score, area under precision-recall curve (AUC-PR) and area under receiver operating characteristic curve (AUC-ROC).

The papers analyzed the different categories of features like Text,language and metadata. Drawbacks of the existing bots like ClueBot and VoABot II were addressed. It was observed that these tools were built around the same primitives that was included in Vandal Fighter. They use lists of regular expressions and consulted databases with blocked users or IP addresses to keep legitimate edits apart from vandalism. The major drawback of these approaches was the fact that these bots utilize static lists of obscenities and 'grammar' rules which are hard to maintain and easy to deceive. As statistics show that they only detected 30% of the committed vandalism. So there was certainly a need for improvement.

It was observed that improvement were achieved by applying machine learning and natural language processing (NLP) techniques. Not in the very least because machine learning algorithms have already proven their usefulness for related tasks such as intrusion detection and spam filtering for email as well as for weblogs. Also most existing works in this area focused on utilizing simple textual features for identifying vandalism. They worked by considering whether an edit contains features that have statistically high likelihood of being associated with vandalism. Examples of such features include abusive/obscene words, spammy words/phrases and certain URLs. These simple approaches, however, have had limited success in combating sophisticated vandal edits often referred to as *elusive vandalism* . These type of vandal attacks are not likely to contain the tell-tale textual features associated with vandalism, and hence they evade common vandalism filters.

A better method was devised to deal with this problem. One paper argued for a *context-aware approach* for detecting vandalism in Wikipedia. The main motivation for considering context was the important observation that the edits in Wikipedia and other CSM applications are not isolated pieces of text. Rather, they happen in a specific *context*. This is in fact a key feature of Wikipedia, and hence it can be highly effective in detecting vandalism. The context of a Wikipedia edit can have multiple distinct aspects such as the relationship of the edit to the article, whether the edit was performed by a registered or an unregistered user, the identity (or the IP address) of the user performing the edit, and the geographical

location from where the edit was performed. The challenge however lied in designing vandalism detection techniques that can effectively harness these various contextual attributes.

Based on these papers a comparative analysis was performed as shown in Table 4.2 below. The table is a summary of the different algorithms that was used in various research papers and jots the advantages and disadvantages of each algorithm.

Table 4.2 Summary of the algorithms

Name	Advantages	Disadvantages
Bayesian Learning	Handles small data limit. Very flexible Interpolates to engineering	Information theoretically problematic. Computationally difficult problems
Decision tree	Well automated. Quite fast	Lack of available memory, when dealing with large databases. Learning problems which can not be solved by decision trees
Neural Network	Able to tolerate noisy data. Successful on several real world application.	Involves long learning time. Require number of parameters that are to be determined empirically.
Support Vector machine	Most robust and accurate methods. Has a sound theoretical foundation.	Computationally expensive. Extremely slow in learning, requiring large amount of training time
Random forest	Not sensitive to noise in the data set. Not subject to over fitting	Observed to over fit for some dataset. Sometimes the variable importance scores are not reliable
Logistic regression	Easily to update model to take in new data . No worries about features being correlated	Requires large sample size to achieve stable results.

Chapter 5

Developing a Wikipedia Vandalism Detection System

5.1 Features

All the features are calculated using metadata and the text of single edits. They can be divided in three groups: Metadata, Text, and Language. Metadata based features are the following:

Anonymous : Whether the editor is anonymous or not.Vandals are likely to be anonymous. This feature is used in a way or another in most antivandalism working bots such as ClueBot and AVBOT. In the PAN-WVC-10 training set (Potthast 2010) anonymous edits represent 29% of the regular edits and 87% of vandalism edits.

Comment length: Length in characters of the edit summary.Long comments might indicate regular editing and short or blank ones might suggest vandalism. However, this feature is quite weak, since leaving an empty comment in regular editing is a common practice.

Size increment: Absolute increment of size, i.e., |new| - |old|.The value of this feature is already well-established since first-generation systems. For example, ClueBot uses various thresholds of size increment for its heuristics, e.g. a big size decrement is considered an indicator of blanking.

Size ratio: Size of the new revision relative to the old revision, i.e., |1+new|/|1+old|. Complements size increment.

Text-based features are the following:

Upper to lower ratio: Uppercase to lowercase letters ratio, i.e., 1+|upper|/1+jlowerj .Vandals often do not follow capitalization rules, writing everything in lowercase or in uppercase.

Upper to all ratio: Uppercase letters to all letters to ratio, i.e., 1+|upper|/1+|lowe|+|upper| .

Digit ratio: Digit to all characters ratio, i.e., 1+|digit|/1+|all| .

This feature helps to spot minor edits that only change numbers. This might help to find some cases of subtle vandalism where the vandal changes arbitrarily a date or a number to introduce misinformation.

Non-alphanumeric ratio: Non-alphanumeric to all characters ratio, i.e., 1+|nonalphanumeric|/1+|all|. An excess of non-alphanumeric characters in short texts might indicate use of emoticons, excessive use of exclamation marks or gibberish.

Character diversity: Measure of different characters compared to the length of inserted text, given by the expression $\text{length}^{1/\text{different char}}$. This feature helps to spot random keyboard hits and other non-sense.

Character distribution: Kullback-Leibler divergence of the character distribution of the inserted text with respect the expectation. Useful to detect nonsense.

Compressibility: Compression rate of inserted text using the LZW algorithm.Useful to detect non-sense, repetitions of the same character or words, etc.

Good tokens: Number of tokens rarely used by vandals, mainly wiki-syntax elements (e.g. __TOC__, <ref>).

Average term frequency: Average relative frequency of inserted words in the new revision.
In long and well-established articles too many words that do not appear in the rest of the article indicates that the edit might be including non-sense or non-related content.

Longest word:Length of the longest inserted word. Its value is 0 if there are no inserted words.
Useful to detect non-sense.

Longest character sequence: Longest sequence of the same character in the inserted Text.

The language-dependent features are based in counters of words in certain categories. For each word category, two features are calculated: frequency and impact. Frequency is the frequency of these words relative to the total words inserted during the edit. Impact is the percentage by which the edit increases the amount of these words. Word categories are :

Vulgarisms: Vulgar and offensive words (eg.shit).

Pronouns: First and second person pronouns, including slang spellings (e.g. I,you, ya).

Bad : Hodgepodge category for colloquial contractions and some typos associated with bad (e.g. wanna, gotcha) and some typos associated with bad writing skills (e.g. dosent).
All: A meta-category containing words from all the previous ones.

Table 5.1: Summary of features

Feature	**Description (Metadata)**
Anonymous	Whether the editor is anonymous or not.
Comment length	Length in characters of the edit summary.
Size increment	Absolute increment of size.
Size ratio	Size of the new revision relative to the old revision.
Feature	**Description (Text)**
Upper to lower ratio	Uppercase to lowercase letters ratio.
Upper to all ratio	Uppercase letters to all letters to ratio.
Digit ratio	Digit to all characters ratio.
Nonalphanumeric	Non-alphanumeric to all characters ratio.

ratio	
Character distribution	KLd between the character distribution of the inserted text and the expectation
Compressibility	Compression rate of inserted text using LZW.
Good tokens	Number of tokens rarely used by vandals, mainly wiki-syntax elements.
Average term frequency	Average relative frequency of inserted words in the new revision.
Longest word	Length of the longest inserted word.
Longest character Sequence.	Longest sequence of the same character in the inserted text.
Feature	**Description (Language)**
Vulgarisms	Vulgar and offensive words.
Pronouns	First and second person pronouns, including slang spellings.
Bad	Hodgepodge category for colloquial contractions and some typos associated with bad and typos associated with bad writing skills.
All	A meta-category containing words from all the previous ones.

5.2 Combining Natural Language, Metadata, and Reputation

5.2.1 Textual and Language features

Text (T): Language-independent features derived from analysis of the edit content. Very long articles may require a significant amount of processing. As the content of the edit is the true guide to its usefulness, there are several ideas for how to measure that property:

Uppercase ratio and digit ratio: Vandals sometimes will add text consisting primarily of capital letters to attract attention; others will change only numerical content. These ratios (and similar ones create features which capture behaviors observed in vandals.

Average and minimum edit quality: Comparing the content of an edit against a future version of the article provides a way to measure the Wikipedia community's approval of the edit .To address the issue of edit warring, the comparison is done against several future revisions. This feature uses edit distance (rather than the blunt detection of reverts) to produce an implicit quality judgement by later edits.

Language

Similar to text features, Language (L) features must inspect edit content. A distinction is made because these features require expert knowledge about the (natural) language. Thus, these features require effort to be re-implemented for each different language. Some of the features included in our analysis are:

Pronoun frequency and pronoun impact : The use of first and second-person pronouns, including slang spellings, is indicative of a biased style of writing discouraged on Wikipedia (non-neutral point-of-view). Frequency considers the ratio of first and second-person pronouns relative to the size of the edit. Impact is the percentage increase in first and second-person pronouns that the edit contributes to the overall article.

Biased and bad words : Certain words indicate a bias by the author (e.g. superlatives: "coolest", "huge"), which is captured by a list of regular expressions. Similarly, a list of bad words captures edits which appear inappropriate for an encyclopedia (e.g. "wanna", "gotcha") and typos (e.g. "seperate"). Both these lists have corresponding frequency and impact features that indicate how much they dominate the edit and increase the presence of biased or bad words in the overall article.

5.2.2 Reputation

We consider a feature in the Reputation (R) category if it necessitates extensive historical processing of Wikipedia to produce a feature value. The high cost of this computational complexity is sometimes mitigated by the ability to build on earlier computations, using incremental calculations.

User reputation : User reputation as computed by WikiTrust . The intuition is that users who have a history of good contributions, and therefore high reputation, are unlikely to commit vandalism.

Country reputation : For anonymous/IP edits, it is useful to consider the geographic region from which an edit originates. This feature represents the likelihood that an editor from a particular country is a vandal, by aggregating behavior histories from that same region.Location is determined by geo-locating the IP address of the editor.

Previous and current text trust histogram : When high-reputation users revise an article and leave text intact, that text accrues reputation, called "trust". Features are-

(1) the histogram of word trust in the edit, and

(2) the difference between the histogram before and after the edit.

5.2.3 Metadata

Metadata (M) refers to properties of a revision that are immediately available, such as the identity of the editor, or the timestamp of the edit. This is an important class of features because it has minimal computational complexity. Beyond the properties of each revision found directly in the database (e.g. whether the editor is anonymous, used by nearly every previous work), there are some examples that we feel expose the unexpected similarities in vandal behaviour.

Time since article last edited : Highly edited articles are frequent targets of vandalism. Similarly, quick fluctuations in content may be indicative of edit wars or other controversy.

Local time-of-day and day-of-week: Using IP geolocation, it is possible to determine the local time when an edit was made. Evidence shows vandalism is most prominent during weekday "school/office hours."

Revision comment length : Vandals decline to follow community convention by leaving either very short revision comments or very long ones.

5.3 Proposed Approach

The main idea is to check how well the words contained in the edit fit into the topic and the existing content of the Wikipedia article. For this, two unique content-based metrics is used to quantifying how compatible an edit is with the context of a Wikipedia article. The first metric, called WWW co-occurrence probability (WCoP) quantifies how often the words in the edit and words in the document appear together (i.e., in the same document) in the corpus of World Wide Web (WWW) documents. The second metric, called top-ranked co-occurrence probability (TCop) is based upon a similar theme, but the corpus is limited to top-ranked (hence, presumably high-quality)WWW documents. The Approach would be as follows-:

For each incoming edit, extract the keywords of the incoming edit and the keywords from the existing version to construct W(E) and W(D) respectively. Using a popular search engine to compute the WCoP and TCop values. These values are fed into machine learning-based classifiers that have been trained on known vandal and nonvandal edit instances. The machine learning-based classifiers determine whether the edit is vandalism. In addition to WCoP/TCop, the machine language-based classifiers utilize more features like Text, Metadata, Language, etc.

5.3.1 Weka

Weka is a popular suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. Weka is free software available under the GNU General Public License.

Weka is a workbench that contains a collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to this functionality. The original non-Java version of Weka was a TCL/TK front-end to (mostly third-party) modeling algorithms implemented in other programming languages, plus data preprocessing utilities in C, and a Makefile-based system for running machine learning experiments. This original version was primarily designed as a tool for analyzing data from agricultural domains,but the more recent fully Java-based version (Weka 3), for which development started in 1997, is now used in many different application areas, in particular for educational purposes and research. Advantages of Weka include: free availability under the GNU General Public License portability, since it is fully implemented in the Java programming language

and thus runs on almost any modern computing platform a comprehensive collection of data preprocessing and modeling techniques
ease of use due to its graphical user interfaces.

Weka supports several standard data mining tasks, more specifically, data preprocessing, clustering, classification, regression, visualization, and feature selection. All of Weka's techniques are predicated on the assumption that the data is available as a single flat file or relation, where each data point is described by a fixed number of attributes (normally, numeric or nominal attributes, but some other attribute types are also supported). Weka provides access to SQL databases using Java Database Connectivity and can process the result returned by a database query. It is not capable of multi-relational data mining, but there is separate software for converting a collection of linked database tables into a single table that is suitable for processing using Weka. Another important area that is currently not covered by the algorithms included in the Weka distribution is sequence modeling.

User interfaces -: Weka's main user interface is the Explorer, but essentially the same functionality can be accessed through the component-based Knowledge Flow interface and from the command line. There is also the Experimenter, which allows the systematic comparison of the predictive performance of Weka's machine learning algorithms on a collection of datasets.

The Explorer interface features several panels providing access to the main components of the workbench: The Preprocess panel has facilities for importing data from a database, a CSV file, etc., and for preprocessing this data using a so-called filtering algorithm. These filters can be used to transform the data (e.g., turning numeric attributes into discrete ones) and make it possible to delete instances and attributes according to specific criteria.
The Classify panel enables the user to apply classification and regression algorithms (indiscriminately called classifiers in Weka) to the resulting dataset, to estimate the accuracy of the resulting predictive model, and to visualize erroneous predictions, ROC curves, etc., or the model itself (if the model is amenable to visualization like, e.g., a decision tree).
The Associate panel provides access to association rule learners that attempt to identify all important interrelationships between attributes in the data.
The Cluster panel gives access to the clustering techniques in Weka, e.g., the simple k-means algorithm. There is also an implementation of the expectation maximization algorithm for learning a mixture of normal distributions.
The Select attributes panel provides algorithms for identifying the most predictive attributes in a dataset.
The Visualize panel shows a scatter plot matrix, where individual scatter plots can be selected and enlarged, and analyzed further using various selection operators.

5.3.2 Word Co-occurrence Probability Matrix

Co-occurrence is a linguistics term that can either mean concurrence / coincidence or, in a more specific sense, the above-chance frequent occurrence of two terms from a text corpus alongside each other in a certain order. Co-occurrence in this linguistic sense can be interpreted as an indicator of semantic proximity or an idiomatic expression. In contrast to collocation, co-occurrence assumes interdependency of the two terms. A co-occurrence restriction is identified when linguistic elements never occur together. Analysis of these restrictions can lead to discoveries about the structure and development of a language.

The overall idea here is to measure the likelihood of the keywords of an incoming edit and the keywords of the existing version of the document occurring together (in the same document) in the World Wide Web (WWW) corpus of documents.
The rationale is that if an incoming edit (represented as E) fits well into the context of the existing version of the Wikipedia page (represented as D), then the keywords of E and D should occur together in a non-negligible fraction of WWW documents.

Let W(D) = {wd1, wd2, . . . , wdn} be the set of keywords in the current (non-vandalized) version of the document. (i.e., W(D) is the current context of the document D) and W(E) ={we1, we2, . . ., wen} denote the set of words that the edit E is seeking to introduce in the next version of the document (i.e.,W(E) is the edit's context). The co-occurrence probability of the arbitrary keyword pair (wei, wdj) is defined as the ratio of the probability that both wei and wdj occur in an arbitrary WWW document to the ratio that at least one of them occurs in a WWW document. Mathematically,

$$CoP(we_i, wd_j) = \frac{P(we_i \in DC \wedge wd_j \in DC)}{P(we_i \in DC \vee wd_j \in DC)}$$

In the above equation, DC denotes an arbitrary WWW document. The denominator in Equation 1 is a normalization term that has been introduced to account for the popularity variations among keywords. The WWW co-occurrence probability is defined as the minimum of the CoPs over all the edit-document keyword pairs.

$$WCoP(E, D) = \underset{we_i \in W(E), wd_j \in W(D)}{\operatorname{argmin}} (CoP(we_i, wd_j)) \quad (2)$$

The reason we use argmin in Equation 2 is that an edit can have only a single vandal word/phrase (i.e., all other words of the edit may be completely legitimate). Thus, we are interested in the contextual fitness (measured by CoP) of the least contextually appropriate word among all the keywords of the edit.

Efficient Estimation Technique:

We need an efficient mechanism for computing the WWW co-occurrence probability metric. The central issue here is to estimate the CoP between various wei-wdj keyword pairs. Our technique for estimating the CoP values works as follows.

A search engine for estimating the CoP values ("Bing" for example). Suppose we want to estimate CoP(wei, wdj). We first issue a search query for documents containing both wei and wej (i.e, the search query will be wei + wdj). Most search engines indicate an estimate on the number of search results (the number of web documents containing both terms). Let the number of search results containing both wei and wdj be represented as Nb. We also issue queries for documents that exclusively contain each one of the search terms.

In other words, we search for (wei – wdj) and (wdj – wei). Let Nei and Nbj be the estimates on the number of search results for these two queries respectively. Now CoP(wei, wdj) is estimated as,

$$\frac{Nb}{(Ne_i+Ne_j+NB)}.$$

An associated problem in computing the WWW cooccurrence probability metric is that the keyword set corresponding to the current version of the document (W(D)) is typically quite large. While edits usually contain a few keywords and phrases, document versions can be quite large. Thus computing CoP values for each edit-document keyword pair becomes prohibitively expensive. This overhead can be alleviated by limiting W(D) to the keywords in the title of the article and its introductory paragraphs. In our experiments , we limit W(D) to the keywords in the document's title.

For example, for text classification where an input document is fed to the model and it should output its class (from a list of classes). The model is trained on many documents with their corresponding classes and when the new document is tested under that model, it will use the features (information) which was extracted from those documents to classify the new document. You will define a vector with fixed length (the number of unique words in your corpus) for each unique word in your corpus. The context vector for each word tells us how many times other words have co-occurred with the current word in the defined window, e.g. in a window of words, you see what are the other words occurred with the current word and increment their corresponding element in the context vector. A simple example is show below :

Corpus: A D C E A D F E B A C E D

Window size: 2 (the 2 words of the either side)

Context vectors:

	A	B	C	D	E
A	0	1	3	2	3
B	1	0	1	0	1
C	3	1	0	2	2
D	2	0	2	0	4
E	3	1	2	4	0

Using these context vectors you can get co-occurrences very easy. For example co-occurrence of D and E is D[E] = 4.

5.4 Implementation Details

For experimental study, Bing search engine will be used (www.bing.com) for calculating the WWW co-occurrence probability and the top-ranked co-occurrence probability. The standard dataset which is used for the project is PAN-WVC-10 corpus. The corpus contains 32452 human-annotated edits on 28468 Wikipedia articles.

Weka 3.7 machine learning toolkit is used for classification and for loading the dataset. The heap size had to be increased in order for this to work. Classifiers which will be used are Naive Bayes, with 10-fold cross validation. Standard language which is used is Java (Eclipse).

5.4.1 Weka steps

These steps show how to use Weka (build **feature vector**, **train** a classifier, **test** a classifier, **use** a classifier) directly from Java code. It is not intended to replace the Explorer/Experimenter GUI that offer the visualization and engineering tools required to set up and debug machine learning experiments. Weka's automation is useful to embed a classifier in a larger program and to create a training/testing loop that can be seen as a regression test for machine learning capabilities. For example,

Step 1: Express the problem with features

This step corresponds to the engineering task needed to write an *.arff* file.

Put all the features in a weka.core.FastVector.

Each feature is contained in a weka.core.Attribute object.

Here, we have two numeric features, one nominal feature (blue, gray, black) and a nominal class (positive, negative).

```
// Declare two numeric attributes
Attribute Attribute1 = new Attribute("firstNumeric");
Attribute Attribute2 = new Attribute("secondNumeric");

// Declare a nominal attribute along with its values
FastVector fvNominalVal = new FastVector(3);
fvNominalVal.addElement("blue");
fvNominalVal.addElement("gray");
fvNominalVal.addElement("black");
Attribute Attribute3 = new Attribute("aNominal", fvNominalVal);

// Declare the class attribute along with its values
FastVector fvClassVal = new FastVector(2);
fvClassVal.addElement("positive");
fvClassVal.addElement("negative");
Attribute ClassAttribute = new Attribute("theClass", fvClassVal);

// Declare the feature vector
FastVector fvWekaAttributes = new FastVector(4);
fvWekaAttributes.addElement(Attribute1);
fvWekaAttributes.addElement(Attribute2);
fvWekaAttributes.addElement(Attribute3);
fvWekaAttributes.addElement(ClassAttribute);
```

Step 2: Train a Classifier

Training requires 1) having a training set of instances and 2) choosing a classifier.

Let's first create an empty training set (weka.core.Instances).

We named the relation "Rel".

The attribute prototype is declared using the vector from step 1.

We give an initial set capacity of 10.

We also declare that the class attribute is the fourth one in the vector (see step 1).

```
// Create an empty training set
Instances isTrainingSet = new Instances("Rel", fvWekaAttributes, 10);
// Set class index
isTrainingSet.setClassIndex(3);
```

Now, let's fill the training set with one instance (weka.core.Instance):

```
// Create the instance
Instance iExample = new DenseInstance(4);
iExample.setValue((Attribute)fvWekaAttributes.elementAt(0), 1.0);
iExample.setValue((Attribute)fvWekaAttributes.elementAt(1), 0.5);
iExample.setValue((Attribute)fvWekaAttributes.elementAt(2), "gray");
iExample.setValue((Attribute)fvWekaAttributes.elementAt(3), "positive");

// add the instance
isTrainingSet.add(iExample);
```

Finally, Choose a classifier (weka.classifiers.Classifier) and create the model. Let's, for example, create a naive Bayes classifier (weka.classifiers.bayes.NaiveBayes).

```
// Create a naïve bayes classifier
Classifier cModel = (Classifier)new NaiveBayes();
cModel.buildClassifier(isTrainingSet);
```

Step 3: Test the classifier

Now that we create and trained a classifier, let's test it. To do so, we need an evaluation module (weka.classifiers.Evaluation) to which we feed a testing set (see section 2, since the testing set is built like the training set).

```
// Test the model
Evaluation eTest = new Evaluation(isTrainingSet);
eTest.evaluateModel(cModel, isTestingSet);
```

The evaluation module can output a bunch of statistics.

```
// Print the result à la Weka explorer:
String strSummary = eTest.toSummaryString();
System.out.println(strSummary);

// Get the confusion matrix
double[][] cmMatrix = eTest.confusionMatrix();
```

Step 4: use the classifier

For real world applications, the actual use of the classifier is the ultimate goal. Here's the simplest way to achieve that. Let's say we've built an instance (named *iUse*) as explained in step 2:

```
// Specify that the instance belong to the training set
// in order to inherit from the set description
iUse.setDataset(isTrainingSet);

// Get the likelihood of each classes
// fDistribution[0] is the probability of being "positive"
// fDistribution[1] is the probability of being "negative"
double[] fDistribution = cModel.distributionForInstance(iUse);
```

5.4.2 Algorithm

The following steps shows how vandalism was detected using a training and a test set by using the supervised version of the filter :-

1) Create data structure and parse it.
2) Set up filter.
3) Set up tokenizer.
4) Use filter.
5) Select attributes.
6) Create classifier to train the model.
7) Create data structure and parse raw data.
8) Set up filter.
9) Set up tokenizer.
10) Prepare test data to labelling.
11) Label test data.

The training set consists of raw data that needs to be parsed and tokenized. Weka selected 35 attributes from the training set to filter data and classify it. Weka used it's own inbuilt classifier- Bayesnet(); for this purpose. A classifier is created to train the model. Again for the test set , data structure is created and raw data is parsed. In the test test , if any id or attribute has something unusual then label it as vandalism. If not then label as normal i.e. corresponding to each user id the user will be labelled as a regular user or a vandalist.

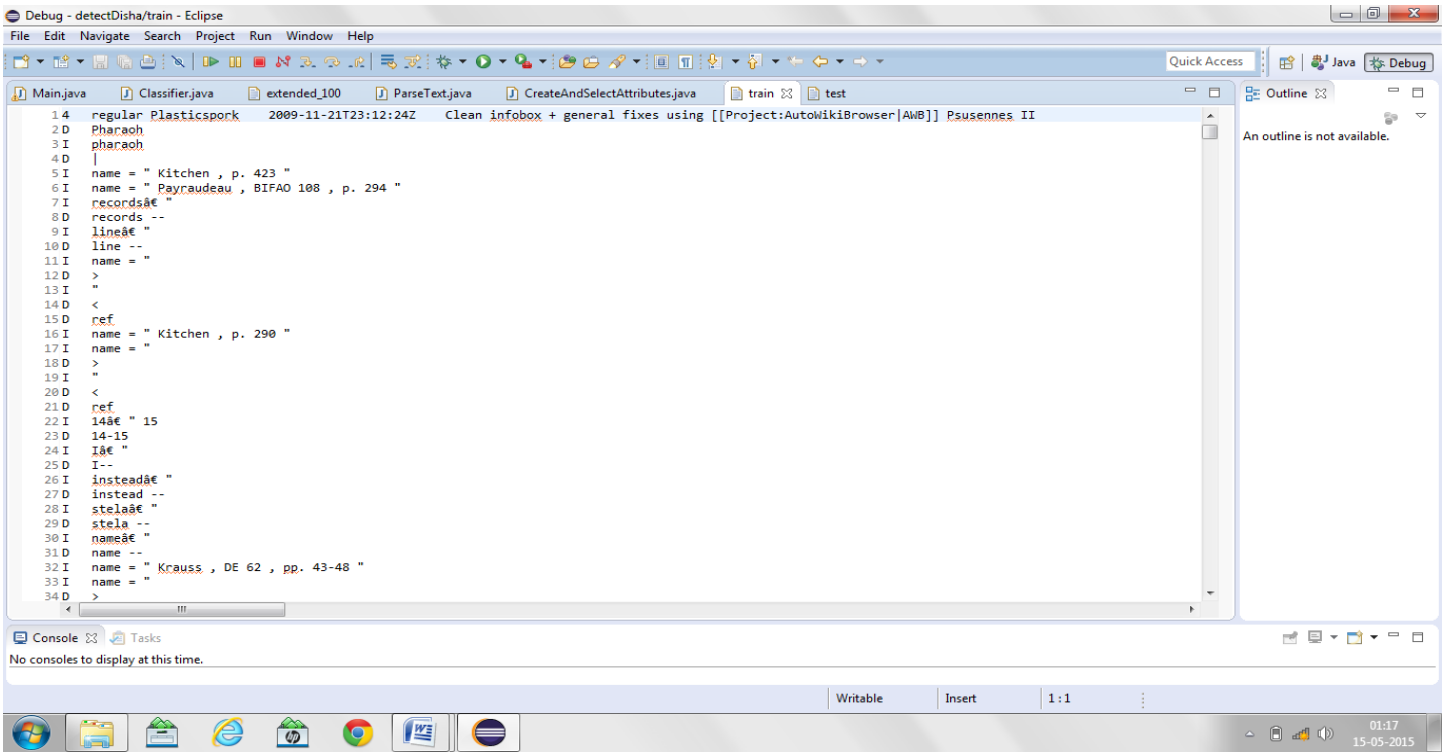
5.4.3 Results

The results consists of console output in which the steps involved are shown. A separate file is generated which has the user id of every user present in the dataset and it labels the user as vandalists or regular user. Weka toolkit was used to improve the efficiency and computation speed which was necessary when a large dataset like the PAN corpus was used.

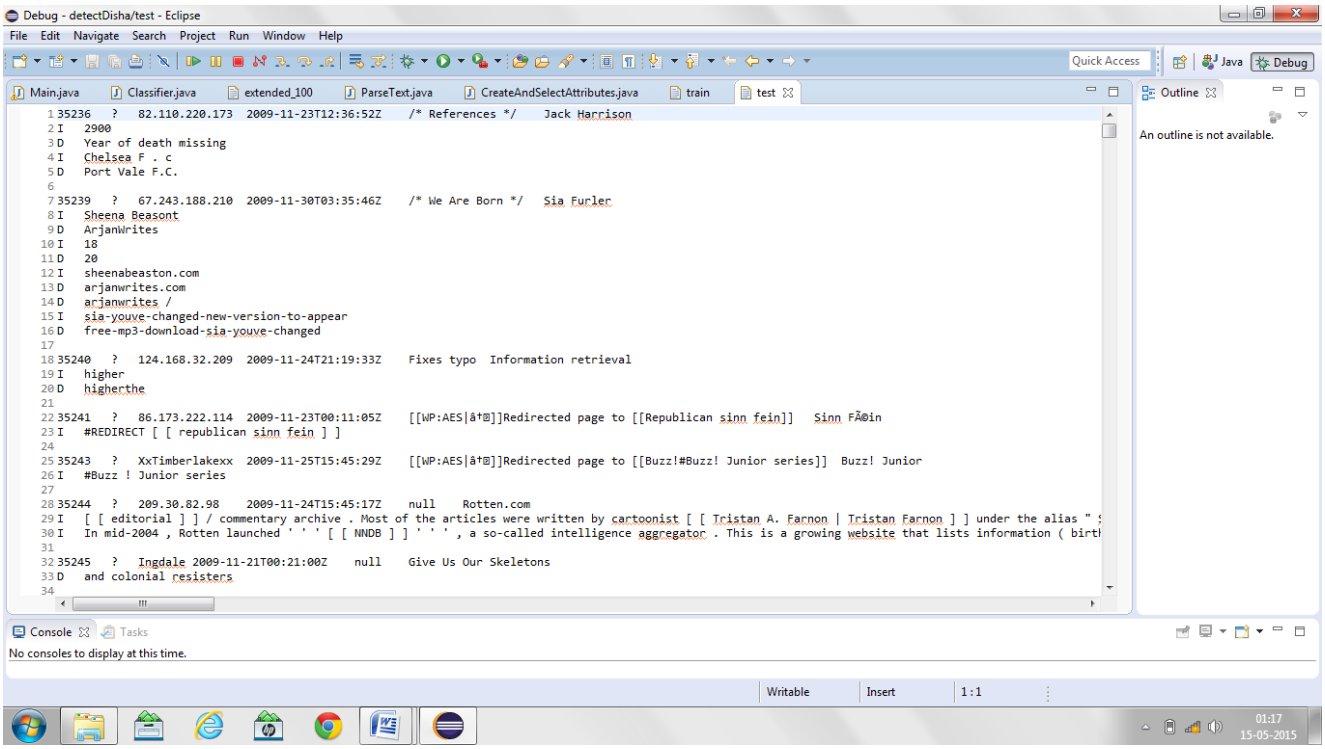
5.4.4 Screenshots

1) Data Set

Training Set

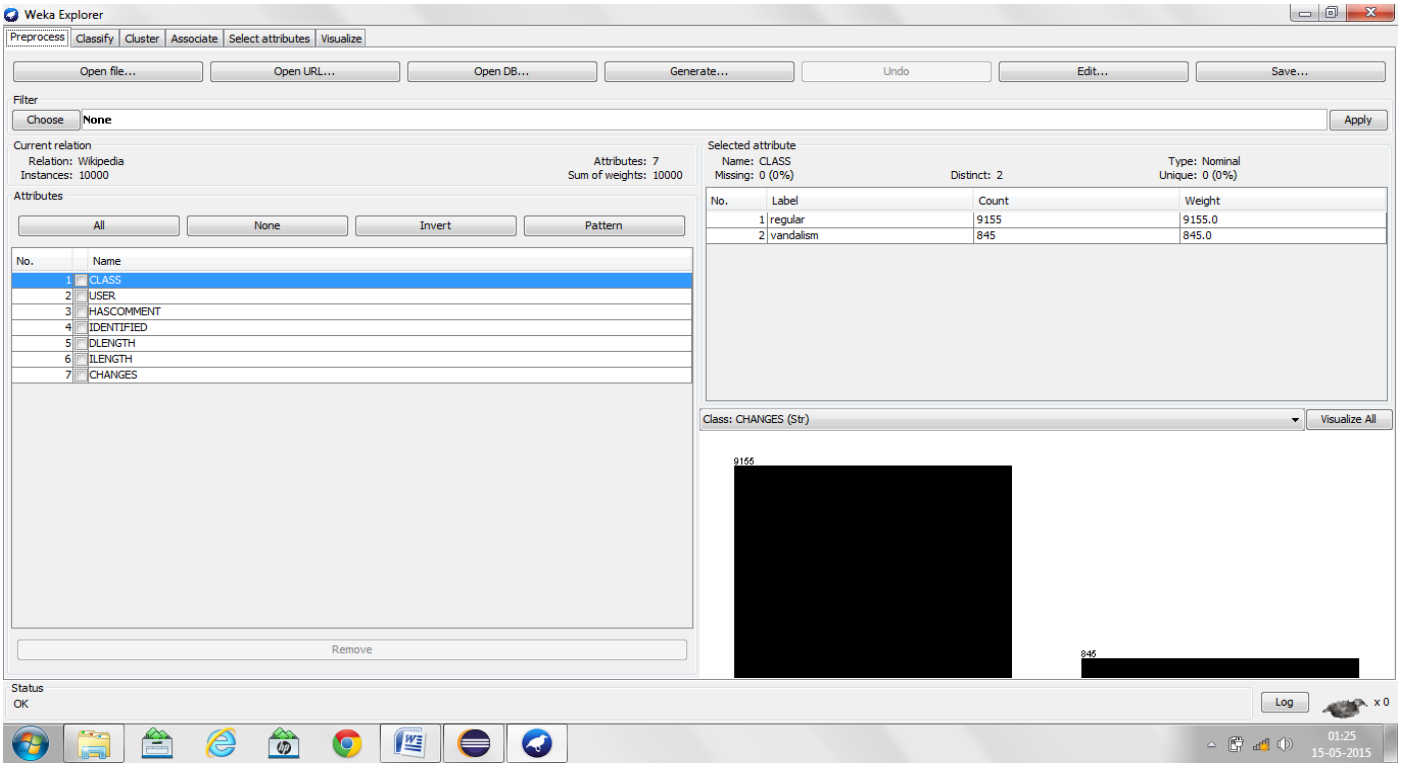


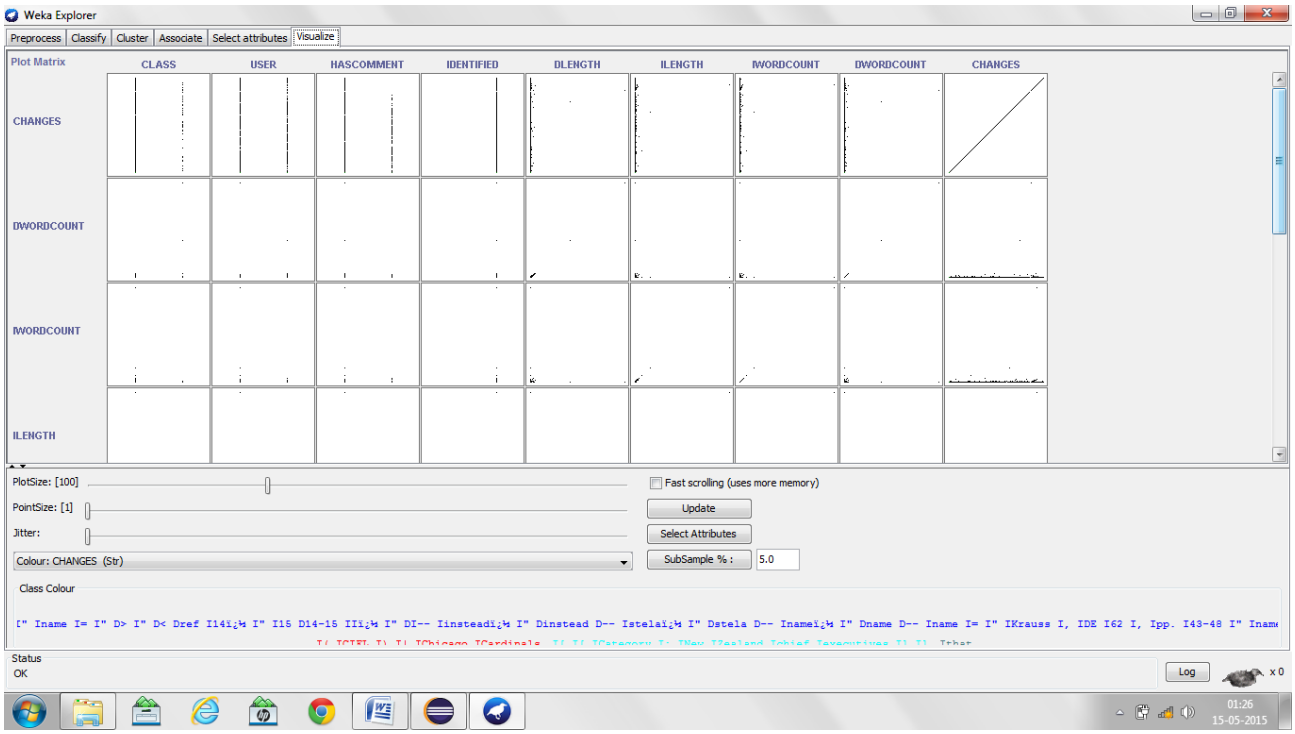
Test Set



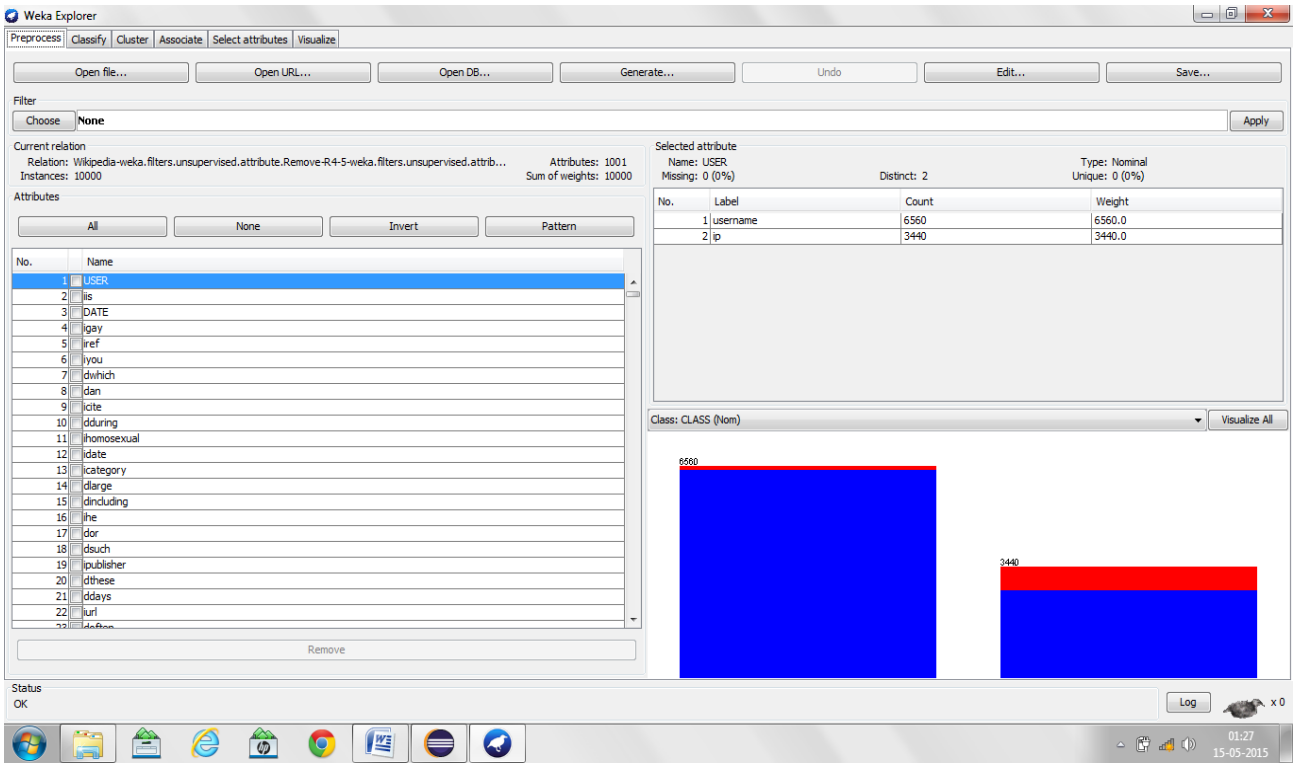
2) Weka Output

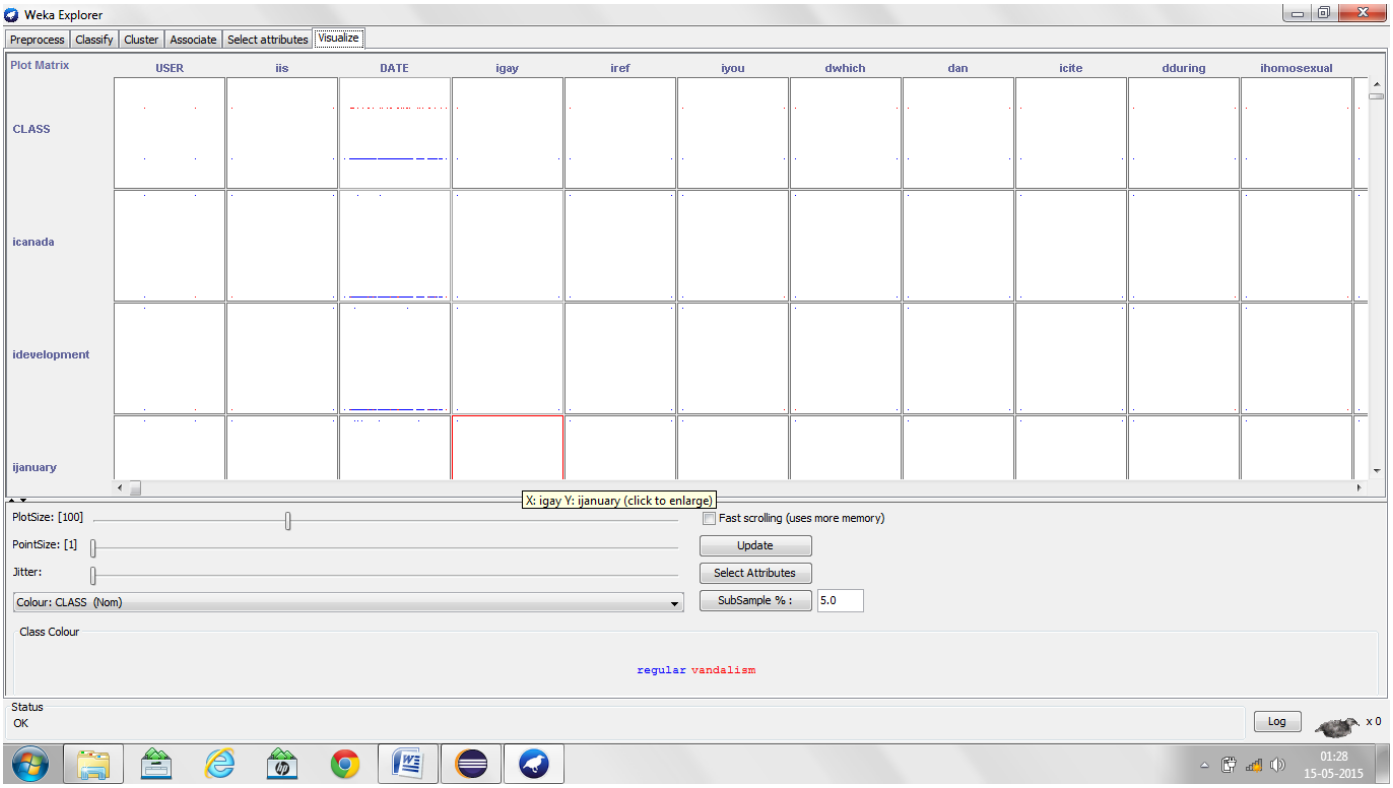
Training arff file



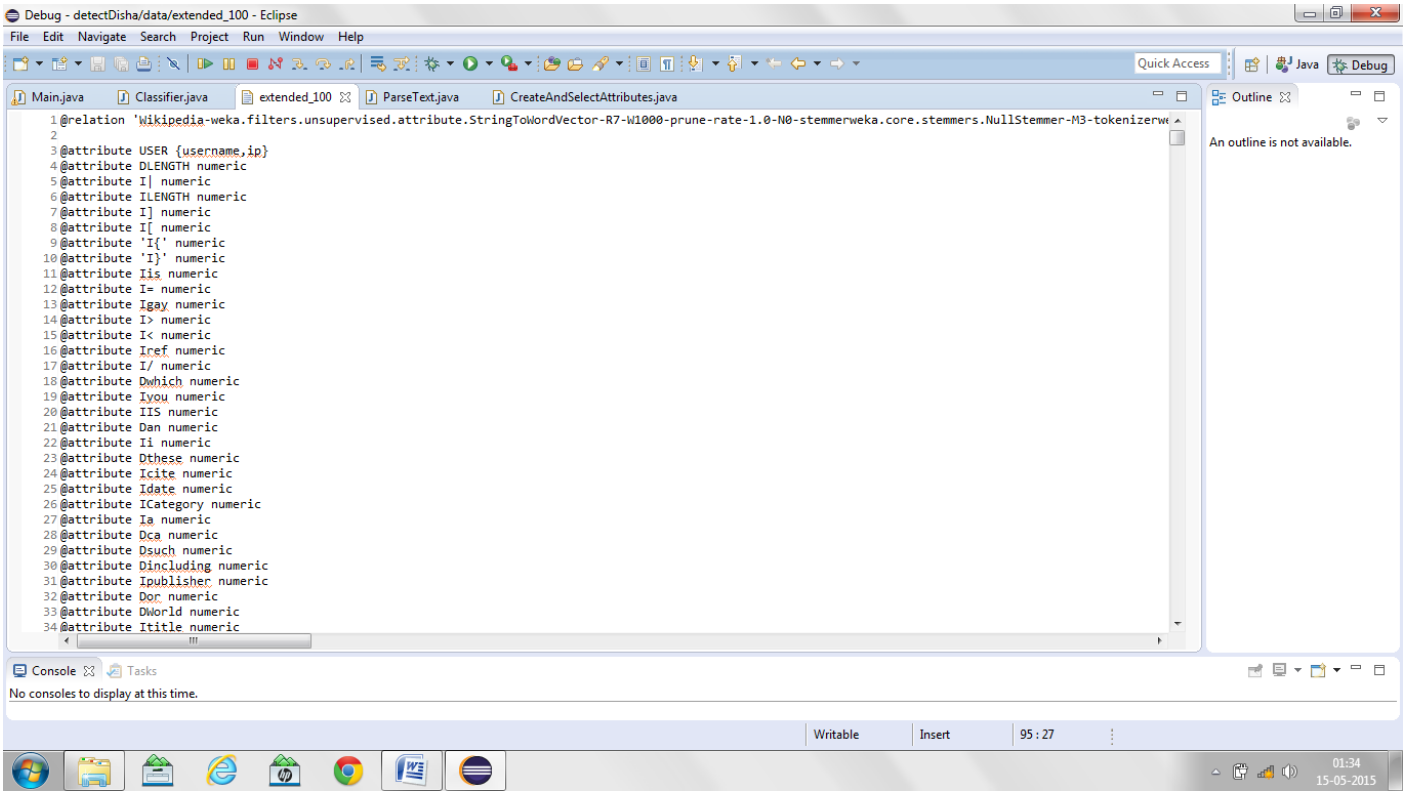


Selected attributes

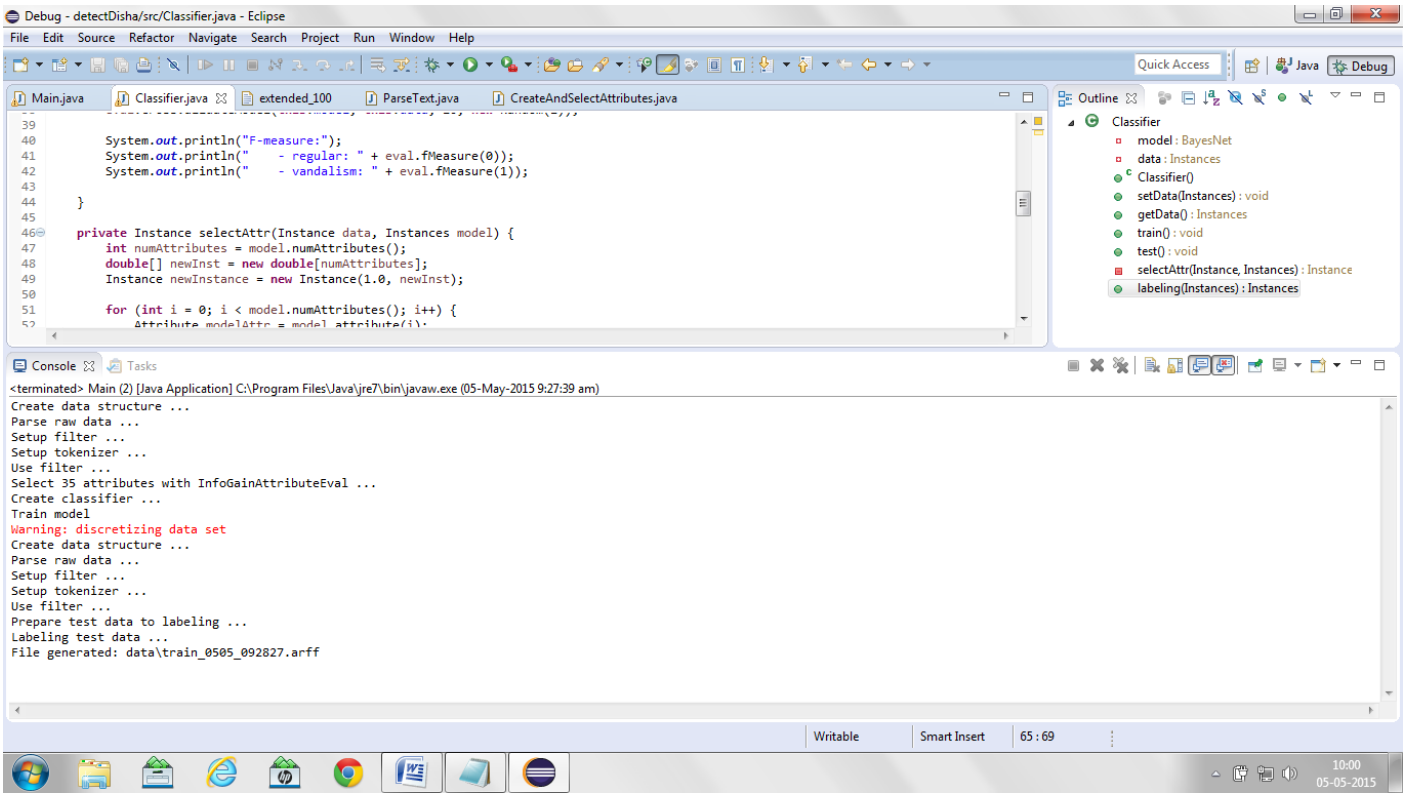




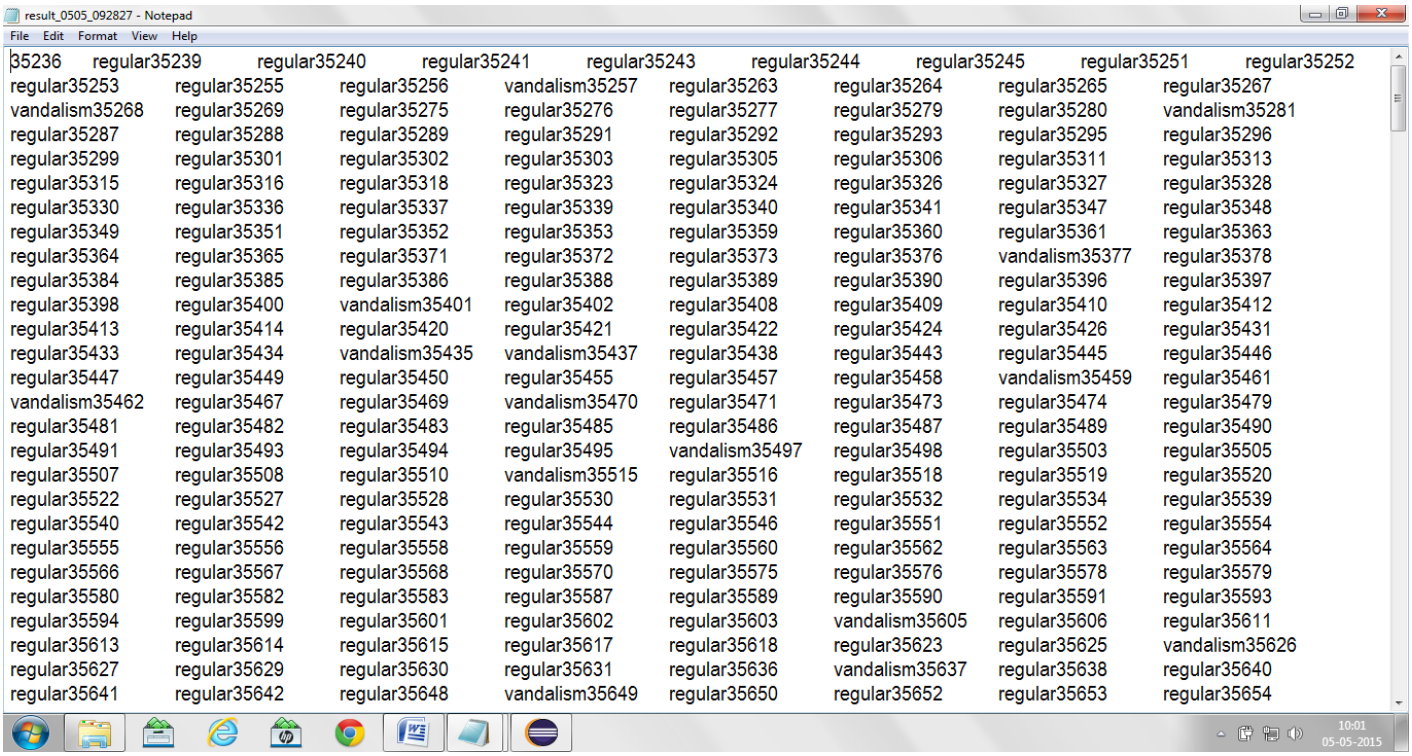
Extended file



3) The console output



4) File generated



Conclusion

Vandalism is a growing problem for Wikipedia and other collaborative social media applications. Vandalism detection techniques that are based upon simple textual features have not been very effective in combating sophisticated vandal attacks.

The success of a machine learning algorithm depends critically on the selection of features that are inputs to the algorithm. Although the previous works on the problem of Wikipedia vandalism detection utilize features from multiple categories, each work has individually focused predominantly on a single category.

In this report, the features of two previous works have been combined.A content-context-centric approach for vandalism detection in Wikipedia is proposed which includes an extended feature set. The main idea is to measure the compatibility of the incoming edit's content with the context of the existing article. Two metrics, namely, WWW co-occurrence probability, has been presented to measure the compatibility of the edit's keywords with the keywords of the existing article.

These features are used in machine learning based classifiers. All the experiments will be performed on Wikipedia vandalism PAN corpus demonstrating that the content-context features significantly improve vandalism detection accuracy when compared with simple textual features.

Appendix

A1 Classifier class

The classifier is built for training an test data. Instances for test data are also labelled.

```
import java.util.Iterator;
import java.util.Random;

import weka.classifiers.Evaluation;
import weka.classifiers.bayes.BayesNet;
import weka.core.Attribute;
import weka.core.FastVector;
import weka.core.Instance;
import weka.core.Instances;

public class Classifier {
	private BayesNet model;
	private Instances data;

	public Classifier() {
		System.out.println("Create classifier ...");
	}

	public void setData(Instances data) {
		this.data = data;
	}

	public Instances getData() {
		return this.data;
	}

	public void train() throws Exception {
		System.out.println("Train model");
		this.model = new BayesNet();
```

```
        this.model.buildClassifier(this.data);
}

public void test() throws Exception {
        System.out.println("Validate model ...");

        Evaluation eval;
        eval = new Evaluation(this.data);
        eval.crossValidateModel(this.model, this.data, 10, new Random(1));

        System.out.println("F-measure:");
        System.out.println("    - regular: " + eval.fMeasure(0));
        System.out.println("    - vandalism: " + eval.fMeasure(1));

}

private Instance selectAttr(Instance data, Instances model) {
        int numAttributes = model.numAttributes();
        double[] newInst = new double[numAttributes];
        Instance newInstance = new Instance(1.0, newInst);

        for (int i = 0; i < model.numAttributes(); i++) {
                Attribute modelAttr = model.attribute(i);
                for (int j = 0; j < data.numAttributes(); j++) {
                        Attribute dataAttr = data.attribute(j);
                        if(modelAttr.equals(dataAttr)) {
                                newInstance.setValue(i, data.value(j));
                        }

                }
        }

        return newInstance;
}

public Instances labeling(Instances unlabeled) throws Exception {
```

```
            FastVector attributes = new FastVector();;

            for (int i = 0; i < this.data.numAttributes(); i++) {
                  attributes.addElement(this.data.attribute(i));
            }

            Instances prepUnlabeled = new Instances("Wikipedia unlabeled", attributes, 0);
            prepUnlabeled.setClassIndex(prepUnlabeled.numAttributes() - 1);



            Instances modelData = new Instances(this.data);
            modelData.delete();

                  System.out.println("Prepare test data to labeling ...");
                  for (int i = 0; i < unlabeled.numInstances(); i++) {
                        Instance element = unlabeled.instance(i);
                        Instance prepared = this.selectAttr(element, modelData);
                        prepUnlabeled.add(prepared);
                  }

            System.out.println("Labeling test data ...");

            for (int i = 0; i < prepUnlabeled.numInstances(); i++) {
                  Instance inst = prepUnlabeled.instance(i);
                  double clsLabel = this.model.classifyInstance(inst);
                  prepUnlabeled.instance(i).setClassValue(clsLabel);
            }

            return prepUnlabeled;
      }

}
```

A2 Parsing data

Here each line of data is read one by one. A data structure is created . Fast vector is used for userLabels,classLabels,Commentlabels,identifiedlabels and attr.

```
import java.util.Iterator;
import java.util.Random;

import weka.classifiers.Evaluation;
import weka.classifiers.bayes.BayesNet;
import weka.core.Attribute;
import weka.core.FastVector;
import weka.core.Instance;
import weka.core.Instances;

public class Classifier {
	private BayesNet model;
	private Instances data;

	public Classifier() {
		System.out.println("Create classifier ...");
	}

	public void setData(Instances data) {
		this.data = data;
	}

	public Instances getData() {
		return this.data;
	}

	public void train() throws Exception {
		System.out.println("Train model");
		this.model = new BayesNet();
		this.model.buildClassifier(this.data);
	}

	public void test() throws Exception {
		System.out.println("Validate model ...");

		Evaluation eval;
```

```
        eval = new Evaluation(this.data);
        eval.crossValidateModel(this.model, this.data, 10, new Random(1));

        System.out.println("F-measure:");
        System.out.println("    - regular: " + eval.fMeasure(0));
        System.out.println("    - vandalism: " + eval.fMeasure(1));

}

private Instance selectAttr(Instance data, Instances model) {
        int numAttributes = model.numAttributes();
        double[] newInst = new double[numAttributes];
        Instance newInstance = new Instance(1.0, newInst);

        for (int i = 0; i < model.numAttributes(); i++) {
                Attribute modelAttr = model.attribute(i);
                for (int j = 0; j < data.numAttributes(); j++) {
                        Attribute dataAttr = data.attribute(j);
                        if(modelAttr.equals(dataAttr)) {
                                newInstance.setValue(i, data.value(j));
                        }

                }
        }

        return newInstance;
}

public Instances labeling(Instances unlabeled) throws Exception {
        FastVector attributes = new FastVector();;

        for (int i = 0; i < this.data.numAttributes(); i++) {
                attributes.addElement(this.data.attribute(i));
        }

        Instances prepUnlabeled = new Instances("Wikipedia unlabeled", attributes, 0);
```

```
            prepUnlabeled.setClassIndex(prepUnlabeled.numAttributes() - 1);


            Instances modelData = new Instances(this.data);
            modelData.delete();

                  System.out.println("Prepare test data to labeling ...");
                  for (int i = 0; i < unlabeled.numInstances(); i++) {
                        Instance element = unlabeled.instance(i);
                        Instance prepared = this.selectAttr(element, modelData);
                        prepUnlabeled.add(prepared);
                  }

            System.out.println("Labeling test data ...");

            for (int i = 0; i < prepUnlabeled.numInstances(); i++) {
                  Instance inst = prepUnlabeled.instance(i);
                  double clsLabel = this.model.classifyInstance(inst);
                  prepUnlabeled.instance(i).setClassValue(clsLabel);
            }

            return prepUnlabeled;
      }

}
```

A3 Creating and selecting attributes

This is done by weka. Word tokenizer and a filter is set here.

```
public class CreateAndSelectAttributes {
      private Instances data;
      private Instances filteredData;
      private Instances selectedAttrData;
      private WordTokenizer tokenizer;
      private StringToWordVector filter;


      public CreateAndSelectAttributes() {
```

```
	}


	public void setData(Instances data) {
		this.data = data;
	}


	private void setupTokenizer() {
		System.out.println("Setup tokenizer ...");
		this.tokenizer = new WordTokenizer();
//		Use default delimiters
//		this.tokenizer.setDelimiters(" \\r\\n\\t.,;:'\"()?!");
	}


	private void setupFilter() throws Exception {
		System.out.println("Setup filter ...");
		this.setupTokenizer();

		this.filter = new StringToWordVector();
//
	this.filter.setOptions(Utils.splitOptions("weka.filters.unsupervised.attribute.StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.NullStemmer -M 1 -tokenizer \"weka.core.tokenizers.WordTokenizer -delimiters \\\" \\\\r\\\\n\\\\t.,;:\\\\\\\\'\\\\\\\\\"()?!\\\\\"\""));
		this.filter.setAttributeIndices("last");
		this.filter.setMinTermFreq(2);
		this.filter.setTokenizer(this.tokenizer);
		this.filter.setStemmer(new NullStemmer());

		this.filter.setInputFormat(this.data);

	}
```

A4 Main class

Objects of other classes are created andthe functions are called. It feteches instances from other classes. A file is generated for the results.

```
ParseText train = new ParseText("train");
        Instances data = train.getInstances();

        CreateAndSelectAttributes CSA = new CreateAndSelectAttributes();
        CSA.setData(data);
        CSA.useFilter();
        CSA.selectAttributes(35);

        data = CSA.getSelectedAttrData();

        Classifier classifier = new Classifier();
        classifier.setData(data);

public static void saveResult(String result) throws IOException {
        String timeStamp = new SimpleDateFormat("MMdd_HHmmss").format(Calendar.getInstance().getTime());

        Writer out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream("result_"+timeStamp+".txt"), "UTF-8"));
        out.write(result);
        out.close();

        System.out.println("File generated: data" + File.separator +"train_"+timeStamp+".arff");
    }


    public static void saveInstances(Instances data) throws IOException {
        String timeStamp = new SimpleDateFormat("MMdd_HHmmss").format(Calendar.getInstance().getTime());

        Writer out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream("data" + File.separator +"train_"+timeStamp+".arff"), "UTF-8"));
        out.write(data.toString());
```

```
            out.close();

            System.out.println("File generated: data" + File.separator +"train_"+timeStamp+".arff");
        }

}
```

References

1. Lakshmish Ramaswamy, Raga Sowmya , Tummalapenta, Kang Li, Calton Pu, "A Content-Context-Centric Approach for Detecting Vandalism in Wikipedia", in 9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2013.
2. Q. Wu, D. Irani, C. Pu, and L. Ramaswamy, "Elusive vandalism detection in wikipedia: a text stability-based approach," in *CIKM*, 2010.
3. Mola-Velasco, S.M.: Wikipedia Vandalism Detection Through Machine Learning:Feature Review and New Proposals. In Braschler, M., Harman, D., eds.: Notebook Papers of CLEF 2010 LABs andWorkshops, 22-23 September, Padua, Italy. (2010).
4. Adler, B., de Alfaro, L., Pye, I.: Detecting Wikipedia Vandalism using WikiTrust. In Braschler, M., Harman, D., eds.: Notebook Papers of CLEF 2010 LABs and Workshops, 22-23 September, Padua, Italy. (2010).
5. West, A.G., Kannan, S., Lee, I.: Detecting Wikipedia Vandalism via Spatio-Temporal Analysis of Revision Metadata. In: EUROSEC'10: Proceedings of the Third European Workshop on System Security. (2010).
6. Potthast, M., Stein, B., Gerling, R.: Automatic Vandalism Detection in Wikipedia.In: ECIR'08: Proceedings of the 30th European Conference on IR Research. Volume 4956 of LNCS., Springer-Verlag (2008).
7. Smets, K., Goethals, B., Verdonk, B.: Automatic Vandalism Detection in Wikipedia: Towards a Machine Learning Approach. In: WikiAI'08: Proceedings of the Workshop on Wikipedia and Arti_cial Intelligence: An Evolving Synergy,AAAI Press (2008).
8. Cristian-Alexandru Drăgușanu, Marina Cufliuc, Adrian Iftene, **"**Detecting Wikipedia Vandalism using Machine Learning": Notebook for PAN at CLEF 2011.
9. B. T. Adler, L. de Alfaro, S. M. Mola-Velasco, P. Rosso, and A. G. West. Wikipedia Vandalism Detection: Combining Natural Language, Metadata, and Reputation Features. In A. Gelbukh, editor,CICLing 2011, volume 6609 of LNCS, Tokyo, Japan,February 2011. Springer.
10. M. Potthast. Crowdsourcing a Wikipedia Vandalism Corpus. In Proc. of the 33rd Intl. ACM SIGIR Conf.(SIGIR 2010). ACM Press, Jul 2010.
11. University of Waikato, "Weka".