Measuring Content Quality in User Generated Content Systems: 
a Machine Learning Approach

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

Sara Javanmardi

Dissertation Committee:
Cristina Videira Lopes, Chair
Donald J. Patterson
David W. McDonald

2011
DEDICATION

To my dear and loving husband Yasser.
Without his continuous support, patience, and most of all love,
the completion of this work would not have been possible.
# TABLE OF CONTENTS

**LIST OF FIGURES**  
vi  
**LIST OF TABLES**  
viii  
**ACKNOWLEDGMENTS**  
x  
**CURRICULUM VITAE**  
xi  
**ABSTRACT OF THE DISSERTATION**  
xiv  

## 1 Introduction  
1.1 Overview  
1.2 Summary of Work and Contributions  
1.3 Dissertation Organization  

## 2 Modeling User Reputation  
2.1 Summary  
2.2 Introduction  
2.3 Background  
2.4 Modeling Reputation  
2.5 Results  
2.6 Tools and Methods  
2.6.1 Extracting Reverts  
2.6.2 Extracting Events  
2.7 Discussion and Comparison to Related Work  
2.8 Conclusions and Future Work  

## 3 Measuring Content Quality  
3.1 Summary  
3.2 Introduction  
3.3 Background  
3.4 Data Set  
3.5 Measuring the Quality Evolution in Articles
3.6 An Empirical Study of Wikipedia Statistics ........................................... 49
  3.6.1 Anonymous vs. Registered Users .................................................. 49
3.7 Conclusion ................................................................. 53

4 Vandalism Detection ...................................................... 57
  4.1 Summary ................................................................. 57
  4.2 Introduction ............................................................... 58
  4.3 Background ............................................................... 60
  4.4 Feature Extraction ........................................................ 65
  4.5 Learning Vandalism Detection Model ............................................. 67
    4.5.1 Vandalism Detection Using Random Forest ................................. 69
  4.6 Feature Selection ........................................................ 71
    4.6.1 Eliminating Groups of Features .............................................. 73
    4.6.2 Eliminating Individual Features ............................................ 75
  4.7 Cost Sensitive Feature Selection ................................................. 80
    4.7.1 Cost Sensitive Lasso ......................................................... 81
    4.7.2 Cost Sensitive Group Lasso .................................................. 82
  4.8 Conclusion ................................................................. 85

5 Handling Class Imbalance Problem ........................................ 88
  5.1 Summary ................................................................. 88
  5.2 Introduction ............................................................... 89
  5.3 Background ............................................................... 90
    5.3.1 Resampling techniques ..................................................... 90
    5.3.2 Cost–sensitive Learning .................................................... 92
    5.3.3 Decision Tree ............................................................... 93
    5.3.4 Random Forest ............................................................. 94
  5.4 Methods ................................................................. 94
  5.5 Handling the Class Imbalance Problem for Random Forests ............ 96
    5.5.1 Seven Configurations ...................................................... 96
    5.5.2 Performance measures ..................................................... 97
    5.5.3 Results ................................................................. 97
  5.6 SMOTE ................................................................. 101
    5.6.1 Map–Reduce on EC2 ....................................................... 103
    5.6.2 Results ................................................................. 104
  5.7 Experiment 3: cost–sensitive Learning ........................................... 107
  5.8 Conclusion ................................................................. 110

6 Handling Missing Data Problem ....................................... 111
  6.1 Summary ................................................................. 111
  6.2 Introduction ............................................................... 112
  6.3 Background ............................................................... 114
  6.4 The Problem of Missing Data ............................................... 116
  6.5 Handling Missing Data ................................................... 117
    6.5.1 Discarding Missing Data .................................................. 118
LIST OF FIGURES

2.1 TPRs and FPRs for the 3 reputation models as the classification threshold is decreased from 1 to 0. 21
2.2 Distribution of reputation for good users/admins vs blocked users based on the 3 models. 22
2.3 Distribution of reputation for all users in English Wikipedia based on the 3 models. 23
2.4 CDR, SDR, DSR and RDR as functions of reputation (based on data before 2007). 26
2.5 CDR, SDR, DSR and RDR extracted after 2007 as functions of reputation computed before 2007. 26
3.1 Transitions between high quality and low quality states 46
3.2 Distribution of Q(T) for featured and non–featured articles 47
3.3 Distribution of QD(T) for featured and non–featured articles 48
3.4 Average article quality for featured articles and non–featured articles. Quality is assessed by the average and the standard deviation of $Q$, $QD$, and $QD50$ for featured and non–featured articles. For each article, $Q$ is the ratio of high–quality revisions. $QD$ is the amount of time that an article spends in its high–quality state computed over its entire lifetime. $QD50$ is the value of $QD$ when only considering the last 50 revisions of the article. 49
3.5 Evolution of article quality over time for same–age articles in Wikipedia. 56
4.1 The effect of number of trees on AUC mean and standard deviation . 72
4.2 Histogram of Cost of Acquisition of Features . 82
5.1 Positives (red) are oversampled by SMOTE to generate synthetic positives (orange) and reinforce minority zones. 91
5.2 Stratified data and C4.5 split: the blue line is a split at a node of the decision tree. Circled in red are negative data cases wrongly put in the positive branch of the tree. 94
5.3 Mean AUC, sensitivity, and specificity for the seven random forest configurations. Both ends of the error bars are one standard deviation distance from the mean. 100
5.4 Average over the 3 folds of the building and testing times for 20 runs of our 7 random forest configurations. 10,000 trees were built. For each run, we only kept 2,000 trees in a random forest and classified a validation fold. .......................................................... 101

5.5 SMOTE on stratified (left) or radial (right) imbalanced data is not always appropriate. Positives (red) oversampled by SMOTE can bring samples that densify the minority zone(s) (orange) or add noise (purple). 102

5.6 Sensitivity plots for oversampling rates of 100 and 1300. ............... 105

5.7 Specificity plots for oversampling rates of 100 and 1300. ............... 106

5.8 AUC against forest size plots for oversampling rates of 100 (top) and 1300 (bottom). .......................................................... 108

6.1 Distribution of user reputation features on train and test sets ......... 121

7.1 A vandalism example that fits in more than one category ............ 132
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>AUC values for the 3 reputation models.</td>
<td>20</td>
</tr>
<tr>
<td>2.2</td>
<td>Mean and standard deviation of reputation values for admins, good users, and blocked users for the 3 reputation models.</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>Correlation values for the 3 reputation models.</td>
<td>26</td>
</tr>
<tr>
<td>2.4</td>
<td>Properties of the data set</td>
<td>27</td>
</tr>
<tr>
<td>2.5</td>
<td>Precision and recall provided by low reputation users for short–lived text.</td>
<td>35</td>
</tr>
<tr>
<td>3.1</td>
<td>Statistical Comparison of Featured and Non–featured Articles. The statistics for featured articles consider revisions submitted before and after the articles were marked as featured</td>
<td>53</td>
</tr>
<tr>
<td>4.1</td>
<td>List of features. Asterisked features are extracted based on both the edit content and its comment.</td>
<td>68</td>
</tr>
<tr>
<td>4.2</td>
<td>Classification performance for the binary classifiers.</td>
<td>69</td>
</tr>
<tr>
<td>4.3</td>
<td>The drop in mean AUC when we eliminate a feature group. Results on train set are for 3–fold cross validation.</td>
<td>74</td>
</tr>
<tr>
<td>4.4</td>
<td>The mean AUC when we only use features in one group.</td>
<td>74</td>
</tr>
<tr>
<td>4.5</td>
<td>Feature selection using lasso. Parameter $\lambda$ determines a trade off between number of selected features and AUC of the classifier. Smaller values of $\lambda$ allow more features be selected and result in models with higher performance.</td>
<td>78</td>
</tr>
<tr>
<td>4.6</td>
<td>Cost Sensitive Feature Selection using Lasso. The lasso path shows how features are added incrementally and the changes in AUC on test set.</td>
<td>83</td>
</tr>
<tr>
<td>4.7</td>
<td>Cost sensitive feature selection using group lasso.</td>
<td>85</td>
</tr>
<tr>
<td>5.1</td>
<td>Paired t–tests results (the table is symmetric).</td>
<td>99</td>
</tr>
<tr>
<td>5.2</td>
<td>Confusion matrix and metrics.</td>
<td>109</td>
</tr>
<tr>
<td>5.3</td>
<td>AUC values for two set of pairs: $C_{FP} = 1$ &amp; $C_{FP} = 10$.</td>
<td>110</td>
</tr>
<tr>
<td>6.1</td>
<td>Correlation analysis between reputation features and class values.</td>
<td>117</td>
</tr>
<tr>
<td>6.2</td>
<td>AUC values for base line: imputing missing values with a unique out-range value</td>
<td>119</td>
</tr>
<tr>
<td>6.3</td>
<td>Discarding instances with missing values from train set.</td>
<td>119</td>
</tr>
<tr>
<td>6.4</td>
<td>Imputing by mean results.</td>
<td>122</td>
</tr>
</tbody>
</table>
6.5 Imputing by mean results. Mean is calculated separately for anonymous and registered users.

6.6 Results for imputing by separate means for anonymous and registered users.

6.7 Imputing by zero results.

6.8 Imputing by learning results.

7.1 Properties of the 5 rounds

7.2 Progress the 5 rounds

7.3 Vandalism Type Distribution Among Consensus Ones

7.4 Vandalism Type Distribution Among Workers’ Answers

7.5 AUC values for the eight vandalism categories

7.6 Classification performance after removing a group of features

7.7 Classification performance after keeping one group of features
ACKNOWLEDGMENTS

I have the deepest gratitude for my advisor, Dr. Cristina Lopes. I would like to thank her for her continued support and encouragement of my academic and professional endeavors. Her insight and encouragement were essential to my personal and professional development. I am greatly indebted to Dr. Pierre Baldi, who invested so much of his time discussing many interesting machine learning topics with me and inspiring me in my research. I especially want to thank Dr. David McDonald for letting me do research in his lab; for assisting me in developing a deep understanding of users needs in the social network system, and for his careful review of my work. I would like to express my gratitude to Dr. Donald Patterson for his review of my dissertation. I would like to acknowledge my gratitude to the faculty of the Donald Bren School of Information and Computer Sciences of California, Irvine. My Special thanks to Dr. Alexander Ihler, Max Welling, Susan Sim, and Gillian Hayes. I would like to thank Thomas Debeauvais and Narges Khatib Astaneh for helping me conduct experiments related to the class imbalance problem. And I would like to thank my family, for their unconditional love and support over these past several years ever since the beginning of my journey. I am especially grateful for my parents, for their encouragement, unwavering confidence and support. To this end, I was remarkably fortunate to become friends with Reza, Rafat, and Arezoo. I wish to thank them or their support and inspiration. There will never be enough time or space to thank all of the people who have helped me reach this point in my life. To those whom I have been unable to mention; please let me offer in a general way my deepest expressions of gratitude. This work has been sponsored in part by the National Science Foundation under grant OCI-074806
CURRICULUM VITAE

Sara Javanmardi

EDUCATION

Ph.D. in Information and Computer Science 2011
University of California, Irvine

M.Sc. in Computer Engineering 2006
Sharif University of Technology

B.Sc. in Computer Engineering 2004
Iran University of Science and Technology

RESEARCH EXPERIENCE

Graduate Research Assistant 2007–2011
University of California, Irvine

Graduate Research Assistant 2005–2006
Sharif University of Technology

TEACHING EXPERIENCE

Teaching Assistant, Information retrieval 2010–2011
University of California, Irvine

PROFESSIONAL EXPERIENCE

University of Washington, Summer 2010
Research Intern, Seattle, WA

Microsoft Research, Summer 2009
Research Intern, Redmond, WA

Microsoft Research, Summer 2008
Research Intern, Redmond, WA
PUBLICATIONS

Finding Patterns in Behavioral Observations by Automatically Labeling Forms of Wikiwork in Barnstars
WikiSym

WikiSym

Trust in Community-Built Databases
Community-Built Databases: Research and Development Book

qSpell: Spelling Correction of Web Search Queries using Ranking Models and Iterative Correction
Microsoft Web Search Workshop

Modeling User Reputation in Wikipedia
Journal of Statistical Analysis and Data Mining

Statistical Measure of Quality in Wikipedia
Microsoft Web Search Workshop

Measuring Quality of the Content in Wikipedia using User Reputation
ICWSM Data Challenge

Scientific Mashups: The Issue of Trust in the Aggregation of Web 2.0 Content
WebScience

Using User Reviews to Improve Search in Wikipedia
SIGIR Workshop on Search in Social Media

Leveraging Crowdsourcing Heuristics to Improve Search in Wikipedia
WikiSym

User Contribution and Trust in Wikipedia
CollaborateCom

Review-based Ranking of Wikipedia Articles
CASoN

CalSWIM: A Wiki-based Data Sharing Platform
CollaborateCom

Modeling Trust in Collaborative Information Systems
CollaborateCom
SELECTED HONORS AND AWARDS

3rd Prize 2011
Microsoft Speller Challenge

Research Grant 2011
Research grant from Amazon.com for working on “Multi–label classification of Wikipedia barnstars”

Research Grant 2011
Research grant from Amazon.com for working on “Distributed Tuning of Machine Learning Algorithms”

3rd Rank 2010
Wikipedia Vandalism Detection

Fellowship and PhD admission 2007
University of California, Irvine
User Generated Content (UGC) has radically transformed the Web from its humble origins as a document–publishing platform. Currently, and most likely in the foreseeable future as well, the Web serves primarily as a social medium, a largely unmoderated platform where millions of people share experiences and knowledge using their own points of view. While this freedom is empowering in general, when left unguided, the Web becomes a cacophony of voices, where fact and fiction, and good information and deception, blur. When faced with poor quality content, users are left with the feeling that nothing on the Web can be trusted.

In order to tackle this issue of trust in unmoderated publishing media, I focus my work on Wikipedia. I set out to devise an efficient mechanism for automatic detection of low quality contributions, commonly known as “vandalism,” and, at the same time, detect contributors who systematically behave as vandals. First I mine the Wikipedia history pages in order to extract user edit patterns. Then I use these patterns to derive several computational models of a user’s reputation. Secondly, based on these models, I generate several new user reputation features and show that they are strong predictors for locating low quality content. To improve the accuracy of my approach,
I extend the feature set by adding other textual features. I describe a method for detecting vandalism that is more accurate than others previously developed.

Because of the high turnaround in user generated content systems, it is important for vandalism detection tools to be scalable and run in real-time. I explain how we can implement the system in a distributed way. In addition, I use cost-sensitive feature selection to reduce the total computational cost of executing our models.

This work is, admittedly, a starting point; but it will prove to be one of great importance if it contributes to a better understanding of user generated content and the methods of measuring and ensuring its quality. The methods I use in this thesis are general and can be applied to numerous other UGCs such as Facebook and Twitter.
Chapter 1

Introduction

1.1 Overview

The advent of user generated content (UGC) marked a shift among media organizations from creating online content to providing facilities for amateurs to publish their own content. UGC content is usually published without any peer review; so one of the big challenges with UGC has to do with the issues of trust and authority.

Many web sites have recognized the importance of creating trust in their community, and they have used users’ reputation systems to leverage their existing systems. For example, good reviews for products on Amazon, established authors in Wikipedia, or eBay sellers with good feedback have proven to greatly influence users’ choices. However, modeling trust in online communities is no simple matter. With each different type of community, we need to recognize when trust is important to its users and what models can be used for creating that trust.

Trust is important when users are either simply learning new information or intending
to make choices based on information. One good example of the former is Wikipedia. As the largest online encyclopedia, it is widely used as a reference. Its articles are invariably among the top search results, consequently enjoying very high visibility. However, compared to traditional web content, there is no notion of authority in Wikipedia. Articles are always subject to changes by unknown users. So it is not guaranteed that the last revision of an article is of a higher quality than the previous ones. Hence, distinguishing between high and low quality has been a concern for Wikipedia since its inception.

Wikipedia has been addressing this issue through different means, such as vandalism detection. According to Wikipedia, vandalism is any addition, removal, or change of content in a deliberate attempt to compromise the integrity of Wikipedia. According to this broad definition, vandalism can include spamming, lobbying, and destroying the edits of others. Wikipedia relies mostly on its human editors and administrators to fight vandalism. But the magnitude of Wikipedia content makes locating all instances of vandalism very time consuming. Tools such as Vandal Fighter, Huggle, and Twinkle are used to monitor recent changes in articles and revert changes deemed vandalistic.

Several machine learning approaches have recently been proposed that would improve vandalism detection. Comparing the performance of these approaches is difficult because of several shortcomings in early vandalism detection corpora. These early corpora were too small, they failed to account for the true distribution of vandalism among all edits, and the hand labeling of the examples had not been double–checked by different annotators. These shortcomings were resolved by the creation of a large–scale corpus for the PAN 2010 competition consisting of a week’s worth of Wikipedia edits.

In this thesis, I study the issue of trust and reputation in UGCs with more focus on the Wikipedia domain. I model user reputation in the context of Wikipedia and develop
a Wiki Reputation management system (WRMS). I show the WRMS is very accurate in assigning reputation values to Wikipedia users. In addition, I show that how user reputation features are strong predictors for measuring content quality. Finally, I use user reputation features in addition to other features in order to develop an efficient and effective vandalism detection system.

1.2 Summary of Work and Contributions

I model vandalism detection as a classification task, for which the goal is to distinguish between vandalistic and legitimate edits. I show the results of my classifier in the PAN 2010 Wikipedia dataset. Using 66 features, I report the AUC of 0.9570 on the PAN test set, which is the best result reported so far. I focus both on effectiveness and efficiency. To build an effective classification model I:

(1) make a rich feature set including strong indicators for detecting vandalistic and non–vandalistic edits. I collect features from different sources. I build upon previous work in spam detection in emails, blogs, and wikis. I also introduce some new features representing user reputation. My feature set includes textual features, meta data features, language model features and user features.

(2) find classification algorithms that maximize the classification performance. To this aim, I run different classifiers while making sure that their free parameters are finely tuned.

(3) handle missing data in some features, such as user reputation features, in order to increase classification performance. In my data set for about 4% of users I do not have enough information to calculate user reputation features. So I need to find robust classifiers that are not sensitive to missing data; or integrate the classification
algorithm with some data imputation methods to alleviate the missing data problem.

(4) handle the class imbalance problem in order to increase the classification performance. Since the data sets are highly skewed and only 7% of the edits are vandalistic, I need to find classifiers that are not sensitive to class skew; or integrate the classification algorithm with some techniques to learn from imbalanced data.

To make the classification task efficient, I:

(1) compact the classification model by removing redundant features. Redundant features are the features that do not contribute to the classification performance because of their irrelevance or strong correlation to other features. (2) build a low–cost classification model where I aim at having a highly accurate classifier that can be acquired at minimal expense. (3) build classifiers whose free parameters can be tuned in an acceptable amount of time. In addition, the decision making phase is fast enough for real time applications. In addition to making the classification scale well, it is highly preferred that both parameter tuning and decision making can be done in a well-distributed manner, for example: based on a MapReduce paradigm.

This work makes three important contributions. First, I provide accurate statistical models in Wikipedia for estimating user reputation based on their edit patterns. Secondly, I use user reputation models to measure content quality. Build upon these, I use machine learning techniques to detect low quality content in Wikipedia in the form of vandalism. While the specific application is to Wikipedia, the machine learning techniques can be generally applied to many forms of UGC. The overall approach is therefore very general and could provide a basis for a wide range of semi–automated vandalism detection tools. To the best of my knowledge, this work provides the best classification results (AUC = 0.9570) on vandalism detection based on PAN 2010 test corpus. Third, one particularly valuable set of features relate to the 'reputation' of
the user. By recognizing the specific value of a reputation function for vandalism detection, we provide a general approach for understanding the cost/benefit trade-off of determining reputation. Further, by illustrating the specific value of these reputation features we raise a critical challenge for large-scale networked data; What are inexpensive techniques for determining user reputation from large complex networked data?

1.3 Dissertation Organization

In chapter 2, I study the issue of trust and reputation among UGCs in general. Then I focus on the domain of Wikipedia and provide three models for estimating user reputation based on their edit patterns. I validate these models using ground-truth Wikipedia data associated with vandals and administrators. When used as a classifier, the best model produces an area under the ROC curve of 0.98. Furthermore, I assess the reputation predictions generated by the models on other users, and show that all three models can be used efficiently for predicting user behavior in Wikipedia.

In chapter 3, I use the reputation models developed in chapter 2 to estimate content quality in Wikipedia. I model the evolution of content quality in Wikipedia articles in order to estimate the fraction of time during which articles retain their high-quality status. The results show that articles tend to have high-quality content 74% of their lifetime and that the average article quality increases as articles go through edits. To further analyze the open editing model of Wikipedia, I compare the behavior of anonymous and registered users and show that there is a positive correlation between registration and the quality of the contributed content. In addition, I compare the evolution of the content in Wikipedia articles that are known to have high-quality content (aka. featured articles) with the rest of the articles in order to extract features
affecting quality. The results show that the high turnover of the content caused by the open editing model of Wikipedia results in rapid elimination of low-quality content. These results suggest two things: first, that the process underlying Wikipedia can be used for producing high-quality content; and secondly, that we might question the viability of collaborative knowledge repositories that impose high barriers to user participation for the purpose of filtering poor quality contributions from the onset.

In chapter 4, I extend the model developed in chapter 3, to locate low-quality content in Wikipedia in the form of vandalism. In addition to user reputation features employed in the previous chapters, I extract other features: textual features, meta data features, and language model features. Based on these features, I describe a method for training classifiers for vandalism detection that yields more accuracy on the PAN 2010 corpus than other classifiers previously developed. Furthermore, because some features are more costly to compute than others, I use cost-sensitive feature selection to reduce the total computational cost of executing my models. In addition to the features previously used for spam detection, I introduce new features based mainly on user action histories. The user reputation features contribute significantly to classifier performance. The approach I use is general and can easily be applied to other user generated content systems.

In chapter 5, I study the problem of class imbalance problems in order to see whether I can improve the classification results reported in chapter 4. Using random forests, I try different resampling and cost-sensitive learning methods. My experiments show that these methods do not improve the classification performance significantly for forests with large number of trees.

In chapter 6, I study the problem of missing data for the user reputation features that are applied to vandalism detection in Wikipedia. I try a variety of methods for handling missing data, based on several binary classifiers, and compare their
suitability and their classification performance. I show that, for some classifiers, like logistic regression, data imputation methods improve classification performance. However, for random forests, imputing missing values with a unique–outrange value yields the best classification results. I show that, on the PAN 2010 corpus, this approach results in AUC=0.9570, which achieves a higher classification performance than all others reported in the previous chapters.

In chapter 7, I report on the construction of the Wikipedia vandalism categories based on the PAN 2010 corpus, using Amazon’s Mechanical Turk. Based on previous literature concerning the different types of vandalism in Wikipedia, I categorize vandalistic edits into eight categories. I explain how I collected the data and discuss the lessons learned. Furthermore, I use binary classifiers to classify vandalism in each of the categories. I measure the classification performance in each category. I show that the classification performance in some categories, such as edit error or comment vandalism, is lower than the rest of the categories. Then, I discuss why this happens and what I can do to improve it.

Finally in chapter 8, I conclude the thesis by discussing the various lessons I have learned from my research and future work.
Chapter 2

Modeling User Reputation

2.1 Summary

Collaborative systems available on the Web allow millions of users to share information through a growing collection of tools and platforms such as wikis, blogs, and shared forums. By their very nature, these systems contain resources and information with different quality levels. The open nature of these systems, however, makes it difficult for users to determine the quality of the available information and the reputation of its providers. Here, we first parse and mine the entire English Wikipedia history pages in order to extract detailed user edit patterns and statistics. We then use these patterns and statistics to derive three computational models of a user’s reputation. Finally, we validate these models using ground–truth Wikipedia data associated with vandals and administrators. When used as a classifier, the best model produces an area under the ROC curve of 0.98. Furthermore, we assess the reputation predictions generated by the models on other users, and show that all three models can be used efficiently for predicting user behavior in Wikipedia.
2.2 Introduction

The last few years have seen a substantial growth in user-generated Web content. Simple editing interfaces encourage users to create and maintain repositories of shared content. Online information repositories such as wikis, forums and blogs have increased the participation of the general public in the production of web content through the notion of social software [89, 5, 14].

Online information repositories, especially in the form of wikis, are widely used on the Web. Wikis are originally designed to hide the association between a wiki page and the authors who have produced it. The main advantages of this feature are: (a) it eliminates the social biases associated with group deliberation, thus contributing to the diversity of opinions and to the collective intelligence of the group, and (b) it directs authors towards group goals, rather than individual benefits [8]. One of the key characteristics of wiki software is its very low-cost collective content creation, requiring only a regular web browser and a simple markup language. This feature makes wiki software a popular choice for content creation projects where minimizing overhead is of high priority; especially in creating new or editing already existing content. For this reason, these platforms can be used in web-based collaborative content management systems for scientific purposes such as team-based research, and e-learning [95, 31, 74].

The most well-known example of a public collaborative information repository is Wikipedia, which has a traffic rank of six worldwide\(^1\). Usually people trust user-generated content in Wikipedia for learning purposes or decision making without validating its information [40]. For these aims, the highly desirable properties of wikis or other similar social software technologies—openness, ease-of-use, and decentralization—

\(^{1}\text{http://www.alexa.com/topsites}\)
can also have disruptive consequences on society. Open wikis can easily be associated with poor-quality information, and often fall prey to malicious or misleading content editing.

Online communities use trust/reputation management components to facilitate cooperative user behavior [104]. In general, trust management systems seek two main goals: (a) to assist users in rating products or other users for better decision making, and (b) to provide an incentive for better user behavior resulting in improved future performance [97, 1]. In the context of wikis, reputation management systems are suggested as a social rewarding technique that motivates users to participate actively in sharing knowledge [54]. In addition, these systems can assist administrators for automatic detection of high/low reputation users to promote/demote the access rights.

*Reputation* can be defined as the opinion (more technically, a social evaluation) of the public toward a person, a group of people, or an organization\(^2\). *Trust* is one user’s belief in another user’s capabilities, honesty and reliability based on his own direct experiences. In online communities, there are two notions of trust: individual-to-individual trust and individual-to-technology trust [25]. eBay and online banking are examples of these two categories, respectively. In Wikis, we have a combination of these trust/reputation relationships; individuals need to have trust in content that is collaboratively created by other individuals. Authors also need to have trust in other authors collaborating with them to create/edit content. For example, one of the obstacles experts who collaborate with Wikipedia face is the lack of guarantee that an inexpert/vandal user will not tamper with their contributed content. Therefore, the trustworthiness of content is tightly linked with the reputation of the author.

In this work, we focus on estimating the reputation of wiki users based on their edit

\(^2\)http://en.wikipedia.org/wiki/Reputation
patterns. We parse and mine entire English Wikipedia history pages in order to extract detailed user edit patterns and statistics. We use these patterns and statistics to derive three computational models of users’ reputation. The main contribution of this work is to accurately infer reputation of users through these mathematical models that are simpler than models previously proposed. With these models, reputation criteria are based on the users’ actions in the system, rather than the explicit, or subjective judgment used in most reputation systems on the Web.

To assess the empirical validity of the models, we evaluate them against some external, independently assessed ground-truth. To do so, we measure the accuracy of the model to determine the reputation of known Wikipedia administrators and vandals. We extend the same experiment to Wikipedia good users and blocked users. Furthermore, we measure the predictive value of the proposed reputation models; we calculate reputation of users in English Wikipedia up to time $t$, then we analyze users’ behavior since then. In aggregate, the results show that the proposed models perform well as classifiers and predictors. Comparison of the model with other similar related work shows that the estimated reputation values are more consistent with users’ behavior.

The remainder of this chapter is as follows: Section 3.3 provides a brief overview of the relevant literature. Section 2.4 introduces the three reputation models. Section 2.5 describes the main results including the validation of the three models. Section 2.6 provides technical background and describes how the Wikipedia data was collected and mined and how the reputation models were implemented. Section 2.7 provides a discussion and compares the present work to related work in the literature. Finally, Section 3.7 draws some conclusions and points to a few directions for future investigation.
2.3 Background

Many online communities have trouble motivating enough users to build an active community. High user participation is the key factor for a successful online community, and that is why good motivating factors are essential [54]. As of August 2011, six of the ten most popular Web sites worldwide simply could not exist without user–contributed content. These sites —Myspace, YouTube, Facebook, eBay, Wikipedia, and Craigslist— look for some incentives to encourage broader participation or the contribution of higher quality content. In order to increase and enhance user–generated content contributions, it is important to understand the factors that lead people to freely share their time and knowledge with others [88, 82].

The positive correlation between content quality and user participation discussed in some work [120, 36]. Some studies also showed that building a good reputation/trust can be a motivating factor that encourages user participation in collaborative systems, as well as an incentive for good behavior [7, 115, 114, 63, 54]. There is an extensive amount of research focused on building trust for online communities through trusted third parties or intermediaries [65, 10]. However, it is not applicable to all online communities where users are equal in their roles and there are no entities that can serve as trusted third parties or intermediaries. Reputation management systems provide a way for building trust through social control without trusted third parties [123].

A reputation management system is an approach to systematically evaluate opinions of online community members on various issues (e.g, products, events, etc.) and their opinions concerning the reputation of other community members [44]. Reputation management systems try to quantify reputation based on metrics to rate their users

---

http://www.alex.com/topsites
or products. In this way, users are able to judge other users or products to decide on future transactions. A well-known example of reputation management is eBay’s auction and feedback mechanism. In this system, buyers and sellers can rate each other after each transaction by crude +1 or −1 values so that the overall reputation of a trustee becomes the sum of these ratings over the last six months. Besides assigning these ratings, users can add textual annotations to present their experiences during their transactions. In other distributed environments such as peer-to-peer (P2P) file sharing networks or grid computing, users can rate each other after each transaction (e.g., downloading a file). So far, a considerable amount of research has been focused on the development of trust/reputation models in virtual organizations, social networks and P2P networks [106, 9, 128, 76].

Reputation management systems are difficult to scale when they have limited sources of information. Users do not always give feedback about other users/products. They also prefer not to return negative feedback [1]. To overcome this problem, these systems consider reputation as a transitive property and try to propagate it, in order to have an estimation of unknown users and products. In this way, there is a high risk of propagating biased, or inaccurate ratings. An study on P2P e-commerce communities confirms this issue and shows that reputation models based solely on feedback from other peers in the community is inaccurate and ineffective [123]. To smooth out this problem, a reputation management system can make its judgments based on objective observations rather than using explicit experiences from other users; for example, by tracking behavior of users in the system, or analyzing users’ feedback to products over time. Quite unlike some research lines that are based on subjective observations in wiki systems [54, 101], in this work we aim at quantifying reputation based on objective observations of the users’ actions.

The idea of mining history revisions in Wikipedia for inferring reliability of the content
was proposed by Zeng et al. [126, 127, 83]. To assess reliability of content, the authors take into account the reputation of the authors regardless of behavior. They categorize users into some groups and assign a static reputation value to each group.

Dynamic reputation estimation to wiki users was discussed in some work [8, 4, 56, 100, 68]. Arazy et al. [8] propose a sentence–ownership algorithm that calculates an author’s contribution to each wiki page. They categorize contributions to wiki pages into four groups: add content, format content, add internal link and add external link. Then, they estimate the extent of contributions each author makes to a wiki page. To assess the accuracy of the proposed algorithm, Arazy et al. compare the author contribution estimated by the proposed algorithm against human judgement for 9 randomly selected articles from Wikipedia. They analyze correlation between the top contributors extracted by their algorithm, and those identified by the human judgement. In summary, the results show that quality of the an author’s contribution to a page is highly correlated with: (a) the number of internal links that the author have added to the page, and (b) the number of sentences contributed by the author that have survived up to the most recent revision of the page.

The most similar work to this study was presented in [4], where authors assign dynamic reputation to users based on their actions. The estimation process is based on the survival of the text, and survival of the edits. In the system, authors gain reputation when their edits are preserved and they lose reputation when their edits are reverted or undone. Because of the vulnerability of the model to some attacks like delete–restore and fake–followers, the reputation estimate algorithm is extended in [18] to prevent such attacks. In this work, we present a robust reputation model that takes into account the survivability of the content. Compared to Adler et al. [4], our model is simpler and the reputation estimate is more accurate. In the next section, we explain the process of modeling user reputation in detail.
2.4 Modeling Reputation

The long-term goal of this effort is to develop an automated system that can estimate the reputation $R_i(t)$ of a Wikipedia user $i$ at time $t$ based on his past behavior. The reputation index $R_i(t)$ should be positive and scaled between 0 and 1 and, for the moment, should be loosely interpretable as the probability that $i$ produces high-quality content. Here we take a first step towards this long term goal by developing several computational models of $R_i(t)$ and testing them, in the form of classifiers, on the available “ground-truth” associated with Wikipedia-known administrators and vandals.

This general approach, which is fairly standard in machine learning applications, requires some explanations. It is reasonable to assume that there exists a true reputation function that is scaled between 0 and 1 and grows monotonically from the user with the lowest reputation to the user with the highest reputation. Our work is an attempt to approximate this unknown function. The only ground-truth available to us concerning this function comes in the form of two extreme datasets of users, the vandals and the admins. No ground-truth data is available for individuals in the middle range of the spectrum. Thus to approximate the true unknown reputation function our first focus is on testing whether the proposed models behave well on the two extreme populations. The models we propose have very few free parameters and they are used to predict reputation values for large numbers of admins and vandals. Once a model capable of producing an output between 0 and 1 for each user has been shown to perform well on the two extreme populations, it is also reasonable to ask whether it performs well on other users. Since no ground truth is available for these users, only indirect evidence can be provided regarding the corresponding performance of the model. Indirect, yet very significant evidence, can be provided in a number of different ways including assessment with respect to other models and
data sets proposed in the relevant literature, and results obtained on curated data sets that go beyond the available admin/vandal data. These are precisely the kind of analyses that are described in the following sections.

In order to estimate users’ reputations, we deconstruct edit actions into inserts and deletes. We consider stability of the inserts done by a user, the fraction of inserts that remain, to be an estimate for his reputation. Although stability of deletes can also be considered as another source of information, it has several shortcomings. In fact, Wikipedia is more derived by inserts, and the size of inserts is 1.6 times larger than the size of deletes. Deletes are more difficult to track and therefore calculating stability of deletes is noisier and more computationally extensive. Hence, we make an assumption that using only stability of inserts would result in a reliable estimation of users’ reputation values.

Consider a user $i$ who at time $t$ inserts $c_i(t)$ tokens into a Wikipedia page. It is reasonable to assume that the update $R_i^{+}(t)$ of $R_i(t)$ should depend on the quality of the tokens inserted at time $t$. To assess the quality of each token, let $t'$ represent the first time point, after $t$, where an administrator (hereafter referred to as “admin”) checks current status of a wiki page by submitting a new revision. According to English Wikipedia history dumps, admins on average submit about 11% of the revisions of pages, which are distributed over the life cycle of the page.

By definition (or approximation), a token inserted at time $t$ is defined to be of good–quality if it is present after the intervention of the admin at time $t'$, otherwise it is considered to be of poor–quality. Therefore we have $c_i(t) = g_i(t) + p_i(t)$ where $g_i(t)$ (resp. $p_i(t)$) represents the number of good–quality tokens (resp. poor–quality). For user $i$, we also let $N_i(t)$ be the total number of tokens inserted up to and right before the time $t$ and, similarly, let $n_i(t)$ be the number of good-quality tokens inserted up to and right before the time $t$. Using a “$+$” superscript to denote values immediately
after the time $t$, we have $N^+_i(t) = N_i(t) + c_i(t)$ and $n^+_i(t) = n_i(t) + g_i(t)$.

We can now define three different models of reputation.

**Model 1:**

In the first model, reputation is simply measured by the fraction of good tokens inserted. In this model, we simply have

$$R^+_i(t) = \frac{n^+_i(t)}{N^+_i(t)} = \frac{n_i(t) + g_i(t)}{N_i(t) + c_i(t)}$$  \hspace{1cm} (2.1)

**Model 2:**

While the first model appears reasonable, tokens that are deleted are treated uniformly. In reality, there is some information to be found in the time at which deletions occur. Vandalistic insertions, for instance, tend to be removed very rapidly [114, 66, 80]. According to our study on Wikipedia, 76% of vandalism is reverted in the very next revision.

Insertions that are deleted only after a very long period of time, tend to be deleted because they are outdated rather than poor in quality. Thus in general, we arrive at the hypothesis that the quicker a token is deleted the more likely it is to be of poor–quality. To realize this hypothesis, we propose a variation on Model 1 where delete tokens introduce a penalty in the numerator with an exponential time decay
controlled by a single parameter $\alpha$.

$$R_i^+(t) = \frac{n_i(t) + g_i(t) - \sum_{d=1}^{p_i(t)} e^{-\alpha(t_d-t)}}{N_i^-(t) + c_i(t)}$$ \hspace{1cm} (2.2)$$

Here $t_d$ represents the time at which the corresponding token was deleted. Since update rate can vary among different wiki pages, we consider the time interval in terms of the number of revisions. We trained $R_i^+(t)$ over different values of $\alpha$ in order to maximize the area under the ROC curve (AUC). The result shows that $\alpha = 0.1$ returns the best result.

Model 3:

this model is a variation of Model 2 where we take into account also the reputation of the deleter, and use his reputation to weigh the corresponding deletion in the form

$$R_i^+(t) = \frac{n_i(t) + g_i(t) - \sum_{d=1}^{p_i(t)} R_{j(t_d)}(t_d)e^{-\alpha(t_d-t)}}{N_i^-(t) + c_i(t)}$$ \hspace{1cm} (2.3)$$

The idea behind this variation of the model is to value the deletions done by high–reputation users (e.g. admins) and devalue the deletions done by low–reputation users (e.g. vandals). In Model 3, $\alpha = 0.08$ for the maximum (AUC).

For users who start with a delete action we need to know the initial value, $R_i(0)$. If we denote $T$ the final time, experiments show that the fastest convergence from $R_i(t)$ to $R_i(T)$ is obtained using the initial values $R_i(0) = 0.2$ for all anonymous users, and $R_i(0) = 0.45$ for all registered users (data not shown). These initial values are used
Finally it is worth noting that if Model 3 were to perform well on the classification task (vandals vs admins) this would provide further indirect evidence that Model 3 is self-consistent and may perform well on other users too, since the update equation at time $t + 1$ for Model 3 uses the predicted reputation for users other than vandals or admins at time $t$.

### 2.5 Results

In this section we evaluate the reputation models on our dataset extracted from English Wikipedia history, as described in Section 5.

#### 2.5.1 Comparison of Models on Ground Truth Data

We first analyze the performance of the reputation models on two major populations: vandals and admins. Vandals are users who have been blocked by the Wikipedia Committee because they performed edits in violation of Wikipedia rules by engaging in vandalism. The ”admin” title is conferred to users selected by the Wikipedia Committee due to their helpful, long-term contributions. Although Model 1, Model 2, and Model 3 have at most one free parameter ($\alpha$) and can be applied directly to estimate the reputation $R_i(t)$ of any user at any time, here we first use the output $R_i(T)$ to derive a classifier to separate vandals and admins. Table 2.1 shows the AUC (Area Under the Curve) values corresponding to the ROC curves of the three corresponding classifiers. The table shows that all the three models perform well and their classification performances are comparable. To further analyze classification performance on a broader set of users, we extend the test populations beyond the
Table 2.1: AUC values for the 3 reputation models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Admins–Vandals</th>
<th>Good Users–Vandals</th>
<th>Admins–Blocked Users</th>
<th>Good Users–Blocked Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1</td>
<td>0.9751</td>
<td>0.9839</td>
<td>0.9196</td>
<td>0.9220</td>
</tr>
<tr>
<td>Model 2</td>
<td>0.9753</td>
<td>0.9769</td>
<td>0.9094</td>
<td>0.9153</td>
</tr>
<tr>
<td>Model 3</td>
<td>0.9742</td>
<td>0.9762</td>
<td>0.9073</td>
<td>0.9125</td>
</tr>
</tbody>
</table>

extreme of vandals and admins to all blocked users on one side and to good users of Wikipedia on the other side.

All blocked users are a superset of the vandals. According to Wikipedia in addition to vandalism, user blocking can happen because of other reasons such as sock-puppetry\(^4\), edit war\(^5\), advertising, or edit disruption. At the other end of the spectrum, automatic extraction of good users beyond admins is not a trivial task. To identify a set of good users, we focus on Wikipedia articles which are marked as good or featured by a committee of experts. From the pool of users contributing to these articles, we extract those who still have contributions that are live in the most recent revisions of these articles. Our definition for good users is also consistent with the result of a recent study of Wikipedia [8], which shows that identification of top page contributors is most highly correlated with the count of their contributed sentences that have survived up to the most recent revision of the wiki pages.

Table 2.1 shows the AUC values for this extended classification experiment. Similar to the previous results, all the three models perform well and their classification performance are comparable; however, looking at TPRs (True Positive Rates) and FPRs (False Positive Rates) separately (Figure 2.1) reveals some subtle differences. In particular, we can see that Model 1 is the best model for detecting vandals/blocked users (lower FPR) while Model 3 is the best model for detecting admin/good users (higher TPR).

\(^5\)http://en.wikipedia.org/wiki/Edit_warring
Figure 2.1: TPRs and FPRs for the 3 reputation models as the classification threshold is decreased from 1 to 0.

Table 2.2: Mean and standard deviation of reputation values for admins, good users, and blocked users for the 3 reputation models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Admins &amp; Good Users</th>
<th>Blocked Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1</td>
<td>0.5416 (±0.2407)</td>
<td>0.0926 (±0.2091)</td>
</tr>
<tr>
<td>Model 2</td>
<td>0.7835 (±0.1698)</td>
<td>0.1884 (±0.2962)</td>
</tr>
<tr>
<td>Model 3</td>
<td>0.8180 (±0.1514)</td>
<td>0.2216 (±0.3128)</td>
</tr>
</tbody>
</table>

Table 2.2 compares the mean and standard deviation of the reputation values for good users and admins against blocked users. In general, all three models assign high reputation values to admins/good users and low reputation values to blocked users; but the distribution of assigned reputations (Figure 2.2) confirms that Model 1 outperforms the other two models at detecting blocked users, while Model 3 outperforms the other two models at detecting good users.
Figure 2.2: Distribution of reputation for good users/admins vs blocked users based on the 3 models.

2.5.2 Reputation and User Behavior

In this section, we consider the application of the three models to estimate the reputation of all users by extending the previous analyses. We first estimate reputation values for all the users of English Wikipedia. Figure 2.3 shows the distribution of reputation values for the three models. Unlike Models 2 and 3, where higher reputation users are more dominant, Model 1 yields a higher number of low reputation users. This is a direct consequence of the prompt punishment of a user in Model 1 after his contributed data is deleted. The decrease in reputation punishment occurs in Model 1 regardless of the reason for the deletion or the reputation of the deleter. Hence, it is very likely that Model 1 overly shifts good users to the left. This is also confirmed by the results of the previous experiments and the poor TPRs of Model 1, compared to Model 2 and Model 3.

In order to evaluate the predictive value of the proposed reputation models, we run another experiment. In this experiment, we calculate the reputation of all the users of English Wikipedia up to time $t$, and analyze the users’ behavior up to time $t$. Then, in a second phase, we analyze their behavior after time $t$, and correlate this behavior with the reputation values calculated before time $t$. Specifically we measure the statistical correlation between the reputation of the users at time $t$ and their behavioral
Figure 2.3: Distribution of reputation for all users in English Wikipedia based on the 3 models.

indicators before and after time $t$. We process history revisions up to January 1, 2007 for reputation estimation, and then examine users’ behavioral indicators on January 1, 2007 and September 30, 2009.

For each model, we classify all the users into 10 different bins (ignoring bots) according to their reputation values. For each bin associated with model, we calculate the mean of four individual, time-dependent, behavioral indicators, namely RDR, DSR, SDR, and CDR, defined as follows:

- **RDR (Reverted Data Ratio)** is the ratio of the number of submitted revisions by a user that are reverted by other users, to the total number of revisions submitted by the same user. This metric can be interpreted as the tendency of a user towards contributing vandalistic/problematic content.

- **DSR (Data Stability Ratio)** is the percentage of contributed data by a user that remains live in the wiki pages. It shows the percentage of content contributed by a user which has not been deleted by other users yet.

- **SDR (Submission Data Ratio)** is the number of revisions submitted by a user to the total number of submitted revisions. This metric shows how actively each user contributes to the wiki pages by submitting new revisions.
• CDR (Correction Data Ratio) is the ration of the number of reverts done by a user to the total number of reverts. This metric can be interpreted as the tendency of a user to make corrections in the wiki pages.

Figure 2.4 shows the mean of CDR, SDR, DSR and RDR respectively, in each bin associated with each reputation model, when the behavioral indicators and reputation values are calculated using data up to January 1, 2007. As the diagrams show, in general, there is a positive correlation between user reputation and CDR – signifying that users with estimated high reputation tend to make more corrections than users with estimated low reputation. The positive correlation between user reputation and SDR also shows that higher reputation users submit more revisions compared to lower reputation users. The correlation between user reputation and RDR is negative, indicating that lower reputation users tend to contribute vandalistic or low quality content more frequently. These positive and negative correlations are consistent with the general intuitions about Wikipedia that were used to build the models.

It is important to note that among these parameters DSR is a direct input to Model 1 and an indirect input to Models 2 and 3. Hence, the positive correlation between DSR and the user reputation is expected for the three models. For this first set of graphs shown in Fig. 2.4 this positive correlation does not give any evidence about the predictive value of the models, since both user behavior indicators and user reputation are calculated on the same data.

To show the predictive value of the models, we plot users’ behavioral indicators computed using data up to September 30, 2009 against the reputation values estimated using data up to January 1, 2007. Figure 2.5 shows the mean of CDR, SDR, DSR and RDR respectively, in each bin associated with each reputation model, where the users’ behavioral indicators are estimated in 2009, while reputation values used to
determine the bins are estimated at the beginning of 2007. The first observation is that this second set of curves has similar shapes to those in Figure 2.4, indicating that the estimated users’ reputations are consistent with their behaviors—users continue to behave in 2007–2009 as they had behaved before 2007. Furthermore, behavior or reputation is captured in broad strokes, by the reputation models.

The values of the behavioral indicators in Figure 2.5 are slightly different from their predicted values corresponding to Figure 2.4. For example, according to Model 3 applied up to 2007, users with a reputation of 0.1 or below ought to have 69% reverted data (RDR), whereas in reality during 2007–2009 those users had only 52% reverted data. Likewise, the same Model 3 predicts that users with a reputation between 0.8 and 0.9 ought to be responsible for 37% of the total number of submissions (SDR), whereas in reality during 2007–2009 those users were responsible for only 27% of submissions. To compare these two sets of diagrams (Figure 2.4 and Figure 2.5), we perform a Pearson correlation analysis. The results are described in Table 2.3, where each tuple shows the correlation between the two parameters before and after 2007, respectively. For example, the entry (−0.906, −0.871) signifies that the correlation between RDR and Model 1 reputation is −0.906 in Figure 2.4, while it is −0.871 in Figure 2.5. These correlations are highly significant and the same is observed in one measures the correlation between the reputation value themselves within or across models, and up to 2007 or up to 2009.

In combination, these results suggest that the reputation models are good at predicting behavioral indices and reputation values at future times, not only for extreme populations of very good of very bad users, but across the entire spectrum of reputation values.
Figure 2.4: CDR, SDR, DSR and RDR as functions of reputation (based on data before 2007).

Figure 2.5: CDR, SDR, DSR and RDR extracted after 2007 as functions of reputation computed before 2007.

Table 2.3: Correlation values for the 3 reputation models.

<table>
<thead>
<tr>
<th>Models</th>
<th>RDR</th>
<th>CDR</th>
<th>SDR</th>
<th>DSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1</td>
<td>(−0.906, −0.871)</td>
<td>(0.434, 0.760)</td>
<td>(0.757, 0.861)</td>
<td>(0.999, 0.996)</td>
</tr>
<tr>
<td>Model 2</td>
<td>(−0.927, −0.939)</td>
<td>(0.783, 0.852)</td>
<td>(0.822, 0.833)</td>
<td>(0.976, 0.975)</td>
</tr>
<tr>
<td>Model 3</td>
<td>(−0.958, −0.973)</td>
<td>(0.779, 0.811)</td>
<td>(0.791, 0.786)</td>
<td>(0.944, 0.944)</td>
</tr>
</tbody>
</table>
Table 2.4: Properties of the data set

<table>
<thead>
<tr>
<th>Time Span</th>
<th>96 months</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Users</strong></td>
<td></td>
</tr>
<tr>
<td>Registered Users</td>
<td>1,749,146</td>
</tr>
<tr>
<td>Anonymous Users</td>
<td>11,048,245</td>
</tr>
<tr>
<td><strong>Number of Articles</strong></td>
<td></td>
</tr>
<tr>
<td>Featured</td>
<td>2,650</td>
</tr>
<tr>
<td>Good Users</td>
<td>197,436</td>
</tr>
<tr>
<td>Good</td>
<td>7,502</td>
</tr>
<tr>
<td>Good Users</td>
<td>334,369</td>
</tr>
<tr>
<td>For Deletion</td>
<td>125</td>
</tr>
<tr>
<td>Regular</td>
<td>1,889,345</td>
</tr>
<tr>
<td><strong>Number of Revisions</strong></td>
<td></td>
</tr>
<tr>
<td>by Anonymous Users</td>
<td>82,577,828</td>
</tr>
<tr>
<td>by Registered Users</td>
<td>41,360,206</td>
</tr>
</tbody>
</table>

2.6 Tools and Methods

In order to get the data for our study, we used 5 client machines for a period of 2.5 months during summer 2009 to send requests to MediaWiki API and extract the data. By sending consecutive requests to MediaWiki API, one can get the text of all revisions of each Wikipedia article. We needed the list of the articles in English Wikipedia to feed to the API in order to get article revisions. However, a significant number of Wikipedia articles had been redirected to other articles which we ignored them. In order to obtain a clean list of Wikipedia articles, we used Crawler4j\(^6\) to crawl English Wikipedia and extract the list of non–redirected articles. We started from the Wikipedia main page and some other seed pages and by traversing the links we crawled about 1.9 million articles. We also used the MediaWiki API to extracted different types of contributors such as *bots*\(^7\), *admins* and blocked users. Table 2.4 shows the properties of the data set.

A note about “users”. It is virtually impossible to associate actual persons with the

\(^6\)http://code.google.com/p/crawler4j/

\(^7\)Bots are generally programs or scripts that make automated edits without the necessity of human decision-making.
internet behavior in a one to one fashion. To bypass this problem Wikipedia defines two classes of users. An anonymous user is a user that is known only through his IP address. A registered user is a user associated with his usernames (i.e. nicknames) that was entered during the registration process. We, as well as others [126, 113, 30], follow the same nomenclature as Wikipedia: a user in this study refers to a registered account or an IP address, and it does not refer to a real–world individual.

2.6.1 Extracting Reverts

A revert is an action to undo all changes made to an article and is primarily used for fighting vandalism. For extracting reverts, we compare the text of each revision to the text of the previous revisions. Since the text comparison process is computationally expensive, the comparison is done on the MD5 signature of the texts rather than on the texts themselves.

2.6.2 Extracting Events

We consider an atomic event to be an insertion or deletion of a word. Insertions are extracted by comparing the text of each revision with the text of the previous revision; deletions are extracted by comparing the text in a revision with the text of the all subsequent revisions. We use the diff algorithm described in [51], for accurate extraction of atomic events. The advantage of this algorithm compared to most of current diff algorithms is its ability to detect movements of blocks. The developed tool, named Wikipedia Event Extractor 8. We calculated $R_i(T)$ of users by processing the extracted events.

---

8http://nile.ics.uci.edu/events-dataset-api/
2.7 Discussion and Comparison to Related Work

In this section, we discuss our model in more detail and compare it to related work in the literature according to several different criteria, appearing in boldface in the list below. criteria:

Tracking Token Ownership.

Effective assignment of inserts and deletes to owners is highly dependent on: (1) the accuracy of the diff algorithm used for calculating the distance between two revisions of a wiki page; (2) the side-effects of reverts resulting in incorrect ownership assignments. An effective diff algorithm for wikis should identify differences in a way that is meaningful to human readers. In particular, reordering of text blocks should be detected in order to accurately assign ownership to the tokens in the reordered blocks. This issue has not been taken into consideration in some of the previous work [101, 52, 8]. For example Sabel et al. [101] use the Levenshtein algorithm\(^9\) to compute the edit distance between two revisions. This algorithm penalizes block reordering and as a result each token that has been shifted is usually considered deleted from its old position, and inserted in its new position [72, 73]. In our experience, the Wikipedia’s diff algorithm can suffer from the same problem, occasionally preventing the detection of block reorderings. We and others [4] overcome this problem by using efficient diff algorithms that detect reordering of blocks and run in time and space linear to the size of the input [51, 110].

Another issue in accurate assignment of token ownership has to do with taking into account the side-effects of reverts. In general, successive revisions of a wiki page have similar content, and each revision, except the very first, is a descendant of the

\(^9\)http://en.wikipedia.org/wiki/Levenshtein_distance
preceding one. However, this model is insufficient for describing the realistic evolution of a wiki page [101]. Assume that a vandal blanks out the \(i\)th revision of a wiki page. Therefore, the \((i + 1)\)th revision becomes blank. When user \(u\) reverts the \((i + 1)\)th revision to the previous revision, this revert results in a new revision and the content of \((i + 2)\)th revision and \(i\)th revision become the same. This scenario raises several problems: (1) users whose contributions were deleted by the vandal are penalized unfairly; (2) \(u\) is erroneously considered to be the owner of all the content of the \((i + 2)\)th revision; (3) the true original owner(s) are denied ownership of the content they actually contributed. We and others [4] address this issue by ignoring these spurious insertions and deletions caused by reverts. However, in [4], the authors decided to process only up to the 3rd successive revision in order to extract reverts and assign ownership. Our study of Wikipedia shows that about 6% of reverts return the \(i\)th revision of a page to the \(j\)th where \(i - j > 3\). For this reason, in order not to lose any information, we process all revisions. Because reverts happen very frequently in Wikipedia, ignoring the side-effect of reverts can result in significant numbers of incorrect assignments of token ownership.

**Stability of Edits.**

For the purpose of this study, user reputation is estimated by looking at the stability of the content he contributes. To estimate the stability of the content, we track the tokens inserted by a user up to the last revision of the page to see how many of these tokens are deleted. In some of the related work in the literature, the tracking process has been more limited, for instance by tracking inserted tokens only up to a limited number of successive revisions and therefore missing some deleted tokens. For example, the authors in [4] use up to the 10th successive revisions. Our study of Wikipedia shows that 37% of the deletes happen after the 10th revision. Hence, ig-
noring this fraction of deletes may lead to reputation estimates that are less accurate. For the purpose of this study user reputation is estimated by considering the stability of inserts only. One may argue that although the number of deletes is considerably smaller than the number of inserts, there is some information in the stability of the deletes too, and one ought to be able to use this additional information to derive even more accurate models of reputation. To see if the stability of deletes can improve the accuracy of the models, we reformulate our simplest model (Model 1) by considering the stability of deletes. We define Model 1’ as follows,

\[
R_i^+(t) = \frac{n_i^+(t) + n_d^+(t)}{N_i^+(t) + N_d^+(t)}
\]

(2.4)

where \(n_d(t)\) is the number of good-quality deleted tokens and \(N_d^+(t)\) is the total number of deleted tokens after time \(t\). We tested Model 1’ as a classifier on admins and vandals and the results showed that Model 1’ has lower AUC (0.84) than Model 1. Interestingly this observation is consistent with the result of another study [8], which shows that delete and proofread edits have little impact on the perception of top contributors in Wikipedia. In other words, there does not seem to exist any significant correlation between an author’s reputation and an author’s number of deletes in the wiki pages; but, in contrast, there is a very strong correlations between an author’s reputation and an author’s number of insertions.

**Dynamic/Non–Dynamic and Individualized/Non–Individualized Reputation Measures.**

One of the advantages of the models presented here is that they assign individualized and dynamic reputation values to both anonymous and registered users. This
is not the case in some of the related work published in the literature. For example, the authors in [126], use non-dynamic and non-individualized reputation values for the users. They categorize users into four groups —administrators, anonymous users, registered users, and blocked users— and assign a static reputation value to each group. In [4], authors consider dynamic and individualized reputation values for registered users, but assign a static and non-individualized reputation value to anonymous users.

**Resistance to Attacks.**

According to the proposed models, users increase their reputation when their contributions to the wiki pages survive. The robustness of the models are highly dependent on when the reputation gain events are triggered. Assume that the reputation of a user increases immediately after he inserts some content; if the page is revised only after a long period of time, the user will have an increased reputation throughout the period, even if his contribution is of poor quality. One solution to this problem is to postpone the reputation increase until the contribution is reviewed by another user. Although this solution solves the previous problem, the reputation model becomes vulnerable to a Sybil attack\(^\text{10}\), in which an attacker has multiple identities and can follow up his own edits. To overcome both problems at once, we postpone the reputation increase until a high-reputation user (e.g. admin) approves the corresponding page. Therefore, in the proposed models, a reputation gain can be triggered only when an admin submits a new revision. One may argue that this reliance on the limited number of admins as outside authorities might reduce the accuracy or scope of applicability of the proposed models. However, as shown in Table2.4, in Wikipedia we have large number of good users which contribute actively to Wikipedia pages.

\(^{10}\)http://en.wikipedia.org/wiki/Sybil\_attack
Thus enlarging the pool of authorities beyond admins to include these good users to validate the quality of insertions may provide an efficient solution, especially for pages with high edit rates.

Among related work, Adler et al. [18] has addressed the attack resistance problem by extending their previously presented model [4]. Although the extended model is resistant to the aforementioned attacks, it is considerably more complex than the original model. Since we do not consider the stability of deletes and reverts and we ignore the side–effect of reverts, our proposed models are not prone to other kinds of attacks, such as delete–restore or fake–followers [18].

Another issue in the proposed models is that reputation gains happen without giving any consideration to the quality of the page that a user contributes to. In [56], the authors make two assumptions: (1) the quality of a wiki page depends on the reputation of its contributors; (2) the reputation of a user depends on the quality of the pages he contributes to. Although the first assumption is often true, the second assumption is more debatable; furthermore, it also increases the vulnerability of the model against some attacks. Our study of Wikipedia shows that vandals are more active in high–quality pages. For example, the average RDR (Reverted Data Ratio) associated with featured articles\(^{11}\) is 17.8% (11.4% before being marked as featured and 25.4% after being marked as featured) while it is about 9.9% for non–featured articles. In general, a policy based on the assumptions in [56], would result in vandals having more incentives to contribute to high–quality pages hoping to increase their reputations, and high reputation users having less incentives to contribute to low–quality pages to improve their quality.

\(^{11}\text{http://en.wikipedia.org/wiki/Featured\_Article}\)
Population Coverage and Precision and Recall Issues.

In related work, anonymous users are either completely ignored or assigned a static reputation value, regardless of their behavior [4]. There are three main reasons why we think that it is important to consider anonymous users in the reputation estimation process: (1) About 33% of the submissions and 39% of the inserts in Wikipedia are contributed by anonymous users and 16% of these contributions have survived up to the last revisions of the articles, therefore they cannot be ignored; (2) Wikipedia itself blocks IP addresses associated with anonymous vandals and 40% of anonymous vandals are subject to infinite blocking. Therefore, an effective reputation management system for Wikipedia should be able to identify anonymous vandals; otherwise, a significant number of vandals will go undetected; and (3) About 15% of data deleted from registered users is deleted by anonymous users, hence ignoring their deletes would degrade the accuracy of the estimated reputation for registered users.

To further verify the relevance of anonymous users, we reformulate Model 3 and assign a static reputation value to all anonymous users, as suggested in [4, 126]. Several static reputation values were tested and the results for the new model (Model 3') show that the AUC always drops, for instance by 1% when the reputation of all anonymous users is set to 0.1. These results indicate that ignoring the anonymous population is likely to decreases the accuracy of a reputation model.

Evaluation results reported by Adler et al [4] using a precision and recall analysis also confirm this observation. To be more specific, in their work they use a model to estimate reputation values up to time $t$, and then estimate the precision and recall after time $t$ provided by low reputation users for short-lived text, which are defined as follows:

- Short-lived text is text that is almost immediately removed (only 20% of the
Table 2.5: Precision and recall provided by low reputation users for short–lived text.

<table>
<thead>
<tr>
<th>Models</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignoring Anonymous Users [4]</td>
<td>0.058</td>
<td>0.378</td>
</tr>
<tr>
<td>Considering Anonymous Users [4]</td>
<td>0.190</td>
<td>0.904</td>
</tr>
<tr>
<td>Model 3</td>
<td>0.404</td>
<td>0.975</td>
</tr>
</tbody>
</table>

text in a version survives to the next version).

- A low-reputation author is an author whose reputation falls in the bottom 20% of the reputation scale.

Table 2.5 shows the precision and recall values obtained on these data by Adler et al by first ignoring anonymous users (first row) and then by assigning a static common reputation value to all anonymous users (second row). The third row shows the results obtained using Model 3, the most similar of our models to their model, to estimate reputations in English Wikipedia up to 2007 and measure precision and recall on the same data. As the table shows, the model by Adler et al [4] performs better when a reputation is assigned to anonymous users, albeit statically. Model 3 significantly outperforms the other two approaches because of dynamic assignment of reputation to anonymous users, better token ownership assignments, and also effective removal of side effects of reverts.

2.8 Conclusions and Future Work

In this study, we have modeled user reputation in wiki systems. We have presented 3 reputation models. According to Model 1, when a user inserts a piece of content in a wiki page his reputation is increased and when a piece of content contributed by the user is deleted by another user, his reputation is decreased. In Model 2, we also take into account the time interval between insertions and deletions of content items.
Finally, in Model 3, we add another parameter which is the current reputation of the deleter.

We have tested the three models on English Wikipedia which is the largest online wiki with an open editing model, allowing anyone to enter and edit content. Our experiments show that the three models can accurately assign reputation values to Wikipedia’s known administrators/good users and vandals/blocked users. Additional analyses reveal that Model 1 does slightly better at detecting vandals and Model 3 does slightly better at detecting good users.

The proposed models can be applied in real time to calculate dynamic and individualized reputation values. While Model 1 is simpler to implement, Model 3 appears to be slightly more accurate, and more robust to attacks than several other competing models of reputation. We are currently exploring several directions for improving the proposed models, for instance, by combining several models (including RDR and DCR) in different ways, and incorporating more information about deletion behavior.

Since all wikis store history pages, the proposed models can be used in any intra-company or intra-organization wikis or in public wikis such as Wikipedia, Citizendium\textsuperscript{12}, and Scholarpedia\textsuperscript{13} and they can be integrated with the corresponding platform. In addition, most of the online wikis like Citizendium and Scholarpedia use the same wiki software, MediaWiki, as Wikipedia. Therefore, the implementation of the models can be modified easily to fit those other platforms. The proposed models can be used in wikis for rating users or as a decision support system for administrators. For example, they can be used for automatic vandal detection, saving substantial amounts of time for wiki administrators. They can be integrated also in a quality assessment system that assesses the reliability of the content according to the reputation of its

\textsuperscript{12}http://en.citizendium.org/
\textsuperscript{13}http://www.scholarpedia.org/
contributors, as suggested in [56].

In the next chapter, we explain how we can use these user reputation values for measuring content quality mainly for the purpose of filtering low–quality content.
Chapter 3

Measuring Content Quality

3.1 Summary

Wikipedia is commonly viewed as the main online encyclopedia. Its content quality, however, has often been questioned due to the open nature of its editing model. A high-quality contribution by an expert may be followed by a low-quality contribution made by an amateur or vandal; therefore the quality of each article may fluctuate over time as it goes through iterations of edits by different users. In this work, we model the evolution of content quality in Wikipedia articles in order to estimate the fraction of time during which articles retain high-quality status. The results show that articles tend to have high-quality content 74% of their lifetime and the average article quality increases as articles go through edits. To further analyze the open editing model of Wikipedia, we compare the behaviour of anonymous and registered users and show that there is a positive correlation between registration and quality of the contributed content. In addition, we compare the evolution of the content in Wikipedia known high-quality articles (aka. featured articles) and the rest of
the articles in order to extract features affecting quality. The results show that the high turnover of the content caused by the open editing model of Wikipedia results in rapid elimination of low–quality content. These results not only suggest that the process underlying Wikipedia can be used for producing high–quality content, but also to question the viability of collaborative knowledge repositories that impose high barriers to user participation for the purpose of filtering poor quality contributions from the onset.

3.2 Introduction

Web 2.0 is the second generation of the web that emphasizes crowdsourcing, the process of outsourcing a task to a large group of people, in the form of an open call [116]. Using wiki technology, Wikipedia has become the largest crowdsourcing project and the main online encyclopedia [129]. It has been suggested that wiki technology can harness the Internet for science; “Wikinomics” is a recent term that denotes the art and science of peer production when masses of people collaborate to create innovative knowledge resources [109]. Because of its open editing model – allowing anyone to enter and edit content – Wikipedia’s overall quality has often been under question. While it is difficult to measure Wikipedia’s overall quality in a definitive way, two studies have tried to assess it manually by comparison of Wikipedia articles to their parallel articles in other reputable sources [40, 23]. Nature magazine’s comparative analysis of forty–two science articles in both Wikipedia and the Encyclopedia Britannica showed a surprisingly small difference; Britannica disputed this finding, saying that the errors in Wikipedia were more serious than the Britannica errors and that the source documents for the study included the junior versions of the encyclopedia as well as the Britannica year books\(^1\).

\(^1\)http://bit.ly/cLDpXO
The questions surrounding Wikipedia’s open editing model have triggered a new generation of wikis like Citizendium\textsuperscript{2} and Scholarpedia\textsuperscript{3}. These online encyclopedias follow a much more traditional editing model, where a small number of experts produce most of the content, through a peer–reviewing process \textsuperscript{4}. However, there is very little evidence that these traditional editing models are better than Wikipedia’s model for the purpose of creating encyclopedic knowledge. To further address these issues, one must develop methods for automatically assessing Wikipedia’s quality and the parameters that affect it.

Since Wikipedia is a highly dynamic system, the articles are changing very frequently. Therefore, the quality of articles is a time–dependent function and a single article may contain high– and low–quality content in different spans of its lifetime. The goal of our study is to analyze the evolution of content in Wikipedia articles over time and estimate the fraction of time that articles are in high–quality state.

This work offers two main contributions to the state of the art. First, we develop an automated measure to estimate quality of article revisions throughout the entire English Wikipedia. Using this measure, we follow the evolution of content quality and show that the fraction of time that articles are in a high–quality state has an increasing trend over time. Then, we present an empirical study of Wikipedia statistics that may explain the results obtained in our study. We analyze the contributions of registered and anonymous users and show that there is a positive correlation between user registration and quality of the contributed content. Furthermore, we compare the evolution of content in featured and non–featured articles over time. We show that featured articles are more closely followed and they benefit from higher content turnover (i.e. higher deletion and replacement of low–quality content).

\textsuperscript{2}http://en.citizendium.org/
\textsuperscript{3}http://www.scholarpedia.org/
\textsuperscript{4}http://bit.ly/9g4yzX
3.3 Background

In the open editing model of Wikipedia users can contribute anonymously or with untested credentials. As a consequence, the quality of Wikipedia articles has been a subject of widespread debate. For example, in late 2005, American journalist John Seigenthaler publicly criticized Wikipedia because of a collection of inaccuracies in his biography page, including an assertion that he was involved with the assassination of former U.S. President John F. Kennedy\(^5\). Apparently the inaccuracies remained in Wikipedia for 132 days. Because there is no single entity taking responsibility for the accuracy of Wikipedia content, and because users have no other way of differentiating accurate content from inaccurate content, it is commonly thought that Wikipedia content cannot be relied upon, even if inaccuracies are rare [26].

To overcome this weakness, Wikipedia has developed several user-driven approaches for evaluating the quality of its articles. For example, some articles are marked as “featured articles”. Featured articles are considered to be the best articles in Wikipedia, as determined by Wikipedia’s editors. Before being listed here, articles are reviewed as “featured article candidates”, according to a special criteria that takes into account: accuracy, neutrality, completeness and style\(^6\). In addition, Wikipedia users keep track of articles that have undergone repeated vandalism in order to eliminate it and report it \(^7\). However, these user-driven approaches cannot be scaled and only a small number of Wikipedia articles are evaluated in this way. For example, as of March 2010, only 2,825 articles (less than 0.1%) in English Wikipedia are marked as featured. Another difficulty of the user-driven evaluations is that Wikipedia content is, by its nature, highly dynamic and the evaluations often become obsolete rather quickly.

\(^7\)http://bit.ly/dy3t1Y
Due to these conditions, recent research work involves automatic quality analysis of Wikipedia [61, 4, 13, 26, 28, 75, 76, 11, 126, 119]. Cross [26] proposes a system of text coloring according to the age of the assertions in a particular article; this enables Wikipedia users to see what assertions in an article have survived after several edits of the article and what assertions are relatively recent and thus, perhaps, less reliable. Adler et al. [3] quantify the reputation of users according to the survival of their edit actions; then they specify ownerships of different parts of the text. Finally, based on the reputation of the user, they estimate the trustworthiness of each word. Javanmardi et al. in [61] present a robust reputation model for wiki users and show that it is not only simpler but also more precise compared to the previous work.

Other research methods try to assess the quality of a Wikipedia article in its entirety. Lih [75] shows that there is a positive correlation between the quality of an article and the number of editors as well as the number of revisions. Liu et. al. [76] present three models for ranking Wikipedia articles according to their level of accuracy. The models are based on the length of the article, the total number of revisions and the reputation of the authors, who are further evaluated by their total number of previous edits. Zeng et al. [126] compute the quality of a particular article revision with a Bayesian network from the reputation of its author, the number of words the author has changed and the quality score of the previous version. They categorize users into several groups and assign a static reputation value to each group, ignoring individual user behavior.

Stvilia et. al. [11] have constructed seven complex metrics using a combination of them for quality measurement. Dondio et al. [28] have derived ten metrics from research related to collaboration in order to predict quality. Blumenstock [13] investigates over 100 partial simple metrics, for example the number of words, characters, sentences, internal and external links, etc. He evaluates the metrics by using them.
for classifications between featured and non–featured articles. Zeng et al., Stvilia et al. and Dondio et al. used a similar method which enables the evaluation results to be compared. Blumenstock demonstrates, with an accuracy of classification of 97%, that the number of words is the best current metric for distinguishing between featured and non–featured articles. These works assume that featured articles are of much higher quality than non–featured articles, and recast the problem as a classification issue. Wohner and Peters [119] suggest that, with improved evaluation methods, these metrics–based studies enable us to determine the accuracy of various submissions. Studying German Wikipedia, they believe that a significant number of non–featured articles are also highly accurate and reliable. However, this category includes a large number of short articles. Their study of German Wikipedia from January 2008 shows that about 50% of the articles contain less than 500 characters, and thereby they assume that some short non–featured articles are of high quality, since their subject matter can be briefly but precisely explained.

In addition, we and others [121, 119] assume that when an article is marked as featured and is displayed on the respective pages, it attracts many more web users for contributions and demands more administrative maintenance. Wohner and Peters’ investigation on German Wikipedia[119] reveals this assumption to be true. For example, over 95% of all articles are edited with greater intensity, once they are marked as featured. Wilkinson and Huberman [121], in a similar study on English Wikipedia, show that featured articles gain an increase in the number of edits and editors after being marked as featured. According to these observations, the accuracy of the classification in the related work ([126, 11, 28]) will be valid only if featured articles are considered before they are marked as featured.
3.4 Data Set

Most of the content analysis research on the evolution of articles (like those enumerated in Section 3.3 and our own work) require the full text of all revisions of articles. We have monitored the publicly available English Wikipedia dumps\(^8\) since early 2006 with the last successful dump released in October 2007. Because of the exponential growth of Wikipedia, all of the history dumps have failed since then. Since the last dump data set is quite out–dated, we created a more recent data set which is now publicly available\(^9\). We used the Wikipedia API\(^{10}\) to get the full text of all the submitted revisions in the history of Wikipedia. The API has a limit of 50 revisions per request and, since these types of requests are not frequent, the chance of having a cached version is slim which makes the process of fetching data expensive. On average, it takes more than one second for the server to send back the result for each request. In addition, we needed to compare the text of subsequent revisions in order to extract the edits made in a revision. This process is also computationally expensive. In order to maintain a reasonable processing speed and still remain polite to Wikipedia servers, we used a cluster of ten nodes which downloaded and processed the whole history of English Wikipedia from July through August 2009. A master node assigned articles to client nodes and waited for them to download and process the article history and send back the extracted statistics.

As of March 2010, English Wikipedia contains about 3.2M articles\(^{11}\). However, some portion of these articles are isolated stubs that are not referenced by any other article. In our analysis, we used crawler4j\(^{12}\) to crawl the entire English Wikipedia and extract

---

\(^8\)http://download.wikimedia.org/enwiki/
\(^9\)http://nile.ics.uci.edu/events-dataset-api/
\(^10\)http://en.wikipedia.org/w/api.php
\(^12\)http://code.google.com/p/crawler4j/
a list of articles accessible through links on the English Wikipedia home page\textsuperscript{13}. We also ignored articles that were redirected to other articles. We ended up with a set of 2.2M articles. Then we downloaded the revisions of these articles through Wikipedia API which resulted in 130M revisions\textsuperscript{14}.

### 3.5 Measuring the Quality Evolution in Articles

Since Wikipedia is a dynamic system, the articles can change very frequently. Therefore, the quality of articles is a time–dependent function and a single content may contain high– and low–quality content in different periods of its lifetime. The goal of our study is to analyze the evolution of content in articles over time and estimate the fraction of time that articles are in high–quality state.

In our analysis of the evolution of the content quality in Wikipedia articles, we divide revisions to low– and high–quality revisions. Based on this assumption, an article can be in low quality \((q = 0)\) or high quality \((q = 1)\) states. In order to assess the quality \(q\) of a revision, we take into account two factors: the reputation of the author and whether this revision has been reverted in one of the subsequent revisions or not. The reputation of a contributor is a value between 0 and 1 and can be viewed as the probability that he produces a contribution of high–quality. This probability is computed based on the stability of the past contributions of the user using the methods developed in [60, 61]. The heuristic behind this reputation assessment is that high–quality contributions tend to survive longer in the articles as compared to low–quality contributions. This heuristic is also supported by other work [56, 4].

As Figure 3.1 suggests, submission of a new revision can keep the state of the article

\textsuperscript{13}http://en.wikipedia.org/wiki/Main_Page

\textsuperscript{14}The dataset is publicly available at http://nile.ics.uci.edu/ events-dataset-api/
or move it to the other state. If the revision is reverted later in the article history, we consider the new state of the article to be $q = 0$. Otherwise, if the reputation of the author of that revision is $r$, then with probability of $r$ the new revision will be $q = 1$ and with probability of $1 - r$ the new revision will be $q = 0$.

With all these elements in place, we define $Q(T)$ as the ratio of high quality revisions submitted for the article up to time $T$:

$$Q(T) = \frac{\sum_{i=1}^{n} q(t_i)}{n}$$  \hspace{1cm} (3.1)

where $q(t_i)$ is the quality of the revision submitted at time $t_i$ and $n$ is the total number of revisions up to time $T$. Figure 3.2 shows the distribution of $Q(T)$ for both all featured articles and a non–featured articles. While the average of $Q(T)$ is relatively high for both featured and non–featured articles, it is higher for featured articles –74% vs. 65%.

To estimate the proportion of time during which an article is in a high–quality state, we also define the duration $QD(T)$ by:
Figure 3.2: Distribution of $Q(T)$ for featured and non–featured articles

\[
QD(T) = \frac{\sum_{i=1}^{n} (t_{i+1} - t_{i})q(t_{i})}{T - t_{1}}
\]  \hspace{1cm} (3.2)

The distribution of $QD(T)$ for both featured and non–feature articles are shown in Figure 3.3. Figure 3.4 also shows the average and standard deviation of $Q(T)$ and $QD(T)$ for both featured and non–featured articles. Featured articles on average contain high–quality content 86% of the time. Interestingly, this value increases to 99% if we only consider the last 50 revisions of the articles. The same statistics...
for non–featured articles show that they have high–quality content 74% of the time. The difference between the averages of $Q(T)$ and $QD(T)$ suggests that typically low–quality content has short life span. This result is consistent with other studies reporting the rapid elimination of vandalism in Wikipedia [113, 66, 80]. For example, [80] reported that about one third to one half of the systematically inserted fictitious claims in Wikipedia are corrected within 48 hours.

Figure 3.5 shows the evolution of $QD(T)$ as a function of $T$ for both featured and non–featured articles of the same age. Overall, $QD(T)$ tends to increase with $T$ and its standard deviation decreases gradually.
Figure 3.4: Average article quality for featured articles and non–featured articles. Quality is assessed by the average and the standard deviation of $Q$, $QD$, and $QD50$ for featured and non–featured articles. For each article, $Q$ is the ratio of high–quality revisions. $QD$ is the amount of time that an article spends in its high–quality state computed over its entire lifetime. $QD50$ is the value of $QD$ when only considering the last 50 revisions of the article.

3.6 An Empirical Study of Wikipedia Statistics

In this section, we present an empirical study of Wikipedia statistics that may explain the results in Section 3.5. First we analyze user attribution in Wikipedia and compare behavior of anonymous and registered users. Second, we compare the evolution of content in featured and non–featured articles to see which parameters result in higher quality in featured articles.

3.6.1 Anonymous vs. Registered Users

Wikipedia users can contribute to wiki pages both anonymously or as registered users. Registered users are identified by their usernames, while anonymous users are tracked by their IP addresses\(^\text{15}\). Although there is no one–to–one correspondence between people and accounts or IP addresses, Wikipedia uses usernames or IP addresses to

\(^{15}\text{http://en.wikipedia.org/wiki/Why_create_an_account}\)
track user behavior for further promotions (e.g. admin assignment) or demotions (e.g. user block). To investigate the effect of open editing model of Wikipedia, we compare the behavior of anonymous and registered users to see if there is any correlation between registration and quality of the contributed content. We, as well as others [126, 113, 30], follow the same nomenclature as Wikipedia: a “user” in this study refers to a registered account or an IP address, and it does not refer to a real-world individual.

Wikipedia keeps the past revisions of articles and these revisions are accessible through history pages of articles. These history pages can be mined in order to analyze the behavior of registered and anonymous users in Wikipedia. Our first attempt to compare the behavior of anonymous and registered users was based on the revert actions done in Wikipedia articles. A revert is the action of undoing all changes made to an article, restoring it to what it was at a specific time in the past. According to the Wikipedia revert policy 16, reversion is used primarily to fight vandalism or similar activities such asspamming. Our study on all English Wikipedia reverts show that 96% of the reverts are done by registered users, while most of the reverted revisions are associated with anonymous users. Furthermore, in 73% of the time a revert restores the current revision of an article to a recent revision submitted by a registered user.

In order to have a more fine-grained analysis of the user behavior, we compared the text of consecutive revisions to extract the insertions and deletions made by each user in each revision. The granularity of inserts and deletes is measured in terms of single tokens (words). The results show that 60.6% of the total inserted content is contributed by registered users. We also followed the evolution of articles and extracted the contributions made to each article over time. Using this method, we

---

16 http://en.wikipedia.org/wiki/Wikipedia:Revert_only_when_necessary
were able to determine the contributor of each single token in the last revision of each article. The results show that 84% of the current content of Wikipedia articles (i.e. survived content in the latest revisions of the articles) has been contributed by registered users. Another interesting observation shows that 49.4% of the contributed content by registered users has been deleted over time while this value is 85.2% for anonymous users. These observations show the high dynamics in the evolution of content in Wikipedia and the higher stability of the registered contributions.

Comparison of the distribution of the reputation for anonymous and registered users clearly shows that registered users tend to have higher reputation. The average of reputation for registered users (as measured in [60, 61]) is 59% while this is 49% for anonymous users [58]. Furthermore, 70% of the reverted revisions (vandalistic content) are associated with anonymous users. Together these results suggest that user registration has a positive affect on the quality of Wikipedia.

**Featured vs. Non–featured Articles**

By comparing content evolution in both featured and non–featured articles, we aim to find what parameters affect content quality and how the open editing model of Wikipedia lets featured articles attain high–quality. In [121, 37], authors have compared featured and non–featured articles and concluded that, on average, featured articles benefit from higher number of edits and distinct editors. In order to have a more detailed comparison between featured and non–featured articles, we examined the evolution of content in these articles and extracted the statistics summarized in Table 3.1.

Though 39.1% of the total inserted tokens in non–featured articles are contributed by anonymous authors, this figure drops to 15.2% when only the last revisions are
considered. In the case of featured articles, the total percentage of inserted tokens by anonymous authors is 56.3%, with this figure dropping to 7.8% in last revisions. According to these statistics, most of the remaining content in both featured and non–featured articles belong to registered users, but this percentage is higher in featured articles. This observation, together with the result of Section 3.6.1 provides strong evidence for why featured articles contain higher quality content throughout their lifetime.

Furthermore, the token survival ratio presented in Table 3.1 shows a much higher turnover of the content in featured articles over time; a higher ratio of tokens is deleted in featured articles compared to other articles. This might be counterintuitive as one might expect that the content inserted in the featured articles should be of higher quality and thereby more stable. However, it can be interpreted as higher dynamics in the evolution of the content in these articles that only allows very high quality content to survive. Note that in order to control the increased visibility and attention that articles might gain after being marked as featured, we have also reported the results of our analysis both before and after becoming featured.

Featured articles can also be distinguished from other articles in terms of proportion of reverted revisions. While on average, 9.9% of the revisions in non–featured articles are reverted, it becomes 25.4% after an article becomes featured. This significant increase in the ratio of reversions after articles are marked as featured is a matter of further study; this can be due to more vandalism as a consequence of higher visibility or it might be attributed to the fact that most of the featured articles have become mature and thus more resistant to change.

In summary, we conclude that (a) featured articles are more closely followed: although less than 0.08% of the articles are marked as featured, they comprise about 1.4% of the total number of revisions; (b) Wikipedia administrators contribute more actively
Table 3.1: Statistical Comparison of Featured and Non–featured Articles. The statistics for featured articles consider revisions submitted before and after the articles were marked as featured.

to featured articles even before these articles are marked as featured; (c) the revert ratio in featured articles is about 1.8 times higher than the ratio for non–featured articles; (d) featured articles have a much higher turnover of content. This higher dynamic in the article’s evolution allows very high quality content to survive. It is interesting to note that even at this lower survival rate, featured articles are on average longer than other articles [13]. Overall, these statistics support the view that featured articles benefit from a higher degree of supervision as compared to other articles.

3.7 Conclusion

Wikipedia is a highly dynamic environment and the quality of its articles can change over time as they go through iterations of edits by different users. In this chapter, we analyzed the evolution of content in English Wikipedia articles and showed that non–featured articles tend to have high–quality content 74% of their lifetime and this is 86% for featured articles. Furthermore, we showed that the average article quality
increases as it go through edits while its standard deviation decreases. To analyze the parameters affecting quality, we compared the behavior of anonymous and registered users and showed that there is a positive correlation between user registration and quality of the contributed content. Furthermore, we examined the evolution of content in both featured and non–featured articles. Higher administrator submissions, higher reversions and higher content turnover are some of the parameters that result in the improvement of featured articles. According to these observations, we can conclude that the open editing model of Wikipedia is not a barrier to quality, its higher user participation rate providing a higher turnover of content, which is, after all, a major prerequisite for high–quality crowdsourcing systems.

In this work, we modeled the quality of an article revision using reputation of the editor and revert status of the revision. Although our study of featured and non–featured articles showed the effectiveness of the model, there are some limitations.

- Data sparsity: for a considerable number of users of Wikipedia we do not have enough information for accurate reputation estimation. The models that we use for estimation of user reputation are based on the observed behavior of users and how other users react to the contributions of these users. Therefore, in cases where a user is new to the system, we do not have a stable reputation estimate for him.

- Anonymity: a significant number of users contribute to Wikipedia articles anonymously and they are identified only by their IP addresses. However, there is a loose correspondence between the IP addresses and the real–word users.

- Expertise: quality of the contribution of a user to a topic depends on the expertise of the user on that topic. Having one reputation value may not be a perfect representative for quality of the contributions of the user on different topics.
In addition to the above limitations, there is no guarantee that users will not change their behavior in the future. So, a user who has contributed high quality content in the past, might contribute low quality content in the future. In addition, when a new user comes to the article and contributes high quality content, the system sacrifices freshness for trustworthiness, only because it does not have an accurate estimate of the user’s reputation. Hence, to estimate content quality, in addition to user reputation we need other features. In the next chapter, we discuss different types of features in addition to user reputation features to estimate content quality. We extend this model by adding different groups of features in order to locate low-quality content in Wikipedia.
Figure 3.5: Evolution of article quality over time for same-age articles in Wikipedia.
Chapter 4

Vandalism Detection

4.1 Summary

A challenge facing user generated content systems is vandalism, edits that damage content quality. The high visibility and easy access to social networks makes them popular targets for vandals. Detecting and removing vandalism is critical for these user generated content systems. Because vandalism can take many forms, there are many different kinds of features that are potentially useful for detecting it. The complex nature of vandalism, and the large number of potential features, make vandalism detection difficult and time consuming for human editors. Machine learning techniques hold promise for developing accurate, tunable, and maintainable models that can be incorporated into vandalism detection tools. In this chapter, we describe a method for training classifiers for vandalism detection that yields classifiers that are more accurate on the PAN 2010 corpus than others previously developed. Moreover, because of the high turnaround in social network systems, it is important for vandalism detection tools to run in real-time, we use feature selection to find the minimal
set of features consistent with high accuracy. In addition, because some features are more costly to compute than others, we use cost-sensitive feature selection to reduce the total computational cost of executing our models. In addition to the features previously used for spam detection, we introduce new features mainly based on user action histories. The user history features contribute significantly to classifier performance. The approach we use is general and can easily be applied to other user generated content systems.

4.2 Introduction

Wikipedia is an open, collaboratively edited encyclopedia that is a heavily used reference source on the Internet. High visibility and the simplicity of editing almost any article have made Wikipedia a popular target for vandals. Maintaining the quality of information in Wikipedia as well as many other User Generated Content (UGC) systems requires identifying and removing vandalism.

In general, vandalism is any deliberate attempt to compromise the integrity of an online source. This covers a broad range of destructive actions such as advertising, attacks on public figures, contributing misinformation, subtle distortion through equivocation or exaggeration, phishing, altering the destination of apparently legitimate links, misleading comments on a change, faulty application of markup text – among many other forms. Really, the range of types of vandalism is quite astonishing.

Many UGC systems, like Wikipedia, rely on extensive manual efforts to combat vandalism. Some automated vandal fighting tools, often in the form of semi-automated bots, are being used to alleviate this laborious task. More recently, machine learning based approaches have been proposed [81]. However, a highly accurate vandalism de-
tection approach that can be applied on the large-scale of Wikipedia is still missing. A successful approach needs to scale well, be robust in the face of widely varying levels of user participation and high user turnover – and should be able to detect vandalism in real-time.

In this chapter we describe the development of a low-cost and highly accurate vandalism detection model that is practical for real-time applications [62]. While our more general focus is on UGC systems, we develop the model based on a Wikipedia corpus from the “Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN)” workshop. This workshop has been developing corpora and testing algorithms head-to-head since 2007 and thus provides data and benchmarks for comparing our results. By aggregating several features used in the PAN competition and identifying a number of new features. We introduce a set of user features which account for past user activity and thereby represent something of the user reputation. The resulting classifier performs better than prior results in the PAN competition.

Further, we try to compress the model by learning from the smallest number of features possible. First, we decrease the feature set by eliminating redundant features. Then, we take into account cost of acquisition of features relative to their individual contribution to the overall performance of the classifier. All the experiments are done based on a MapReduce paradigm, which makes our approach both efficient and scalable.

This work makes two important contributions. First, while our specific application is to Wikipedia, the machine learning techniques are focused on leveraging low-cost features that can be generally applied to many forms of UGC. The overall approach is therefore very general and could provide a basis for a wide range of semi-automated vandalism detection tools. Second, one particularly valuable set of features relate to the ‘reputation’ of the user. By recognizing the specific value of a reputation
function for vandalism detection, we provide a general approach for understanding the cost/benefit trade–off of determining reputation. Further, by illustrating the specific value of these reputation features we raise a critical challenge for large–scale networked data; What are inexpensive techniques for determining user reputation from large complex networked data?

This chapter is structured in the following way. We begin with a review of the relevant work on vandalism mitigation from both a user and a technical perspective. Through this we identify a number of persistent challenges for users as well as sets of features that are commonly used to develop technical solutions for vandalism detection. In subsequent sections we elaborate a relevant set of features, and use those features to train a classifier that is effective at predicting vandalism. We then apply lasso to learn a sparse low–cost model whose accuracy is comparable to the original classification model. We close the chapter by considering some applications of the resulting model and the more general implications of our approach and findings.

4.3 Background

Vandalism detection has been a concern for Wikipedia since its inception. Vandalism in Wikipedia is defined as any addition, removal, or change of content in a deliberate attempt to compromise the integrity of Wikipedia. According to this broad definition, vandalism can include spamming, lobbying and destroying the edits of others. Wikipedia relies mostly on its human editors and administrators to fight vandalism; identifying instances of potential vandalism by reading a diff of the change and reverting changes that look to be a form of vandalism. But the scale of Wikipedia makes locating all vandalism very labor intensive. Tools such as Vandal Fighter, Huggle, and Twinkle are used to monitor recent changes to articles in real–time and revert
those changes deemed vandalism [38].

In the following we describe two broad approaches to vandal fighting; (a) the user approach which relies mostly on tools to assist user detection, and (b) more automated approaches that generally rely on bot or other algorithms.

Viegas et al. [114] conducted some early work on the types of vandalism found in Wikipedia. They used a visualization technique called “History Flow” to see the various ways pages were edited and changed over time. In considering these changes they identified five types of vandalism: Mass Deletion, Offensive Copy, Phony Copy, Phony Redirection, and Idiosyncratic Copy. They also analyzed the time for repair of vandalism, which they termed “survival time”. They found that the median survival time for some vandalism is quite short, on the order of minutes. However, they noted that the mean time is skewed long, on the order of days or weeks, depending on the type of vandalism. This means there is some vandalism that goes undetected for long periods of time.

In some follow–up work, Priedhorsky et al. [92], considered the impact of a piece of vandalism. That is, if some vandalism lasts on a site for days, how likely is it that a user might stumble across that and be misinformed or otherwise get a wrong impression about the quality of the entire content based on a vandalized page. They developed a model based on page viewing behaviors and vandalism persistence. Their model correlates closely with the empirical results from Viegas et al. [114] and further illustrates that about 11% of vandalism would last beyond 100 page views, with a small fraction of a percent lasting beyond 1000 page views.

Priedhorsky et al. [92] also considered the types of vandalism and how likely users were to agree on the types of vandalism. They began with the vandalism types from [114] making some small refinements and identified two additional types: Misinforma-
tion, and Partial Delete. They identified the most frequent categories of vandalism as Nonsense (Phony Copy) 53%, Offensive 28%, Misinformation 20% and Partial Delete 14%. However, they also noted that the rate of agreement among their users for some categories is somewhat low with Misinformation having the lowest level of agreement for these top four categories. This means that some of the most subtle vandalism, in the form of Misinformation, is even hard for users to agree upon.

In recent work, Geiger & Ribes [38] studied the process of editors who participate in “Recent Changes Patrolling” using Huggle. Their study raises some nice issues about the work to remove vandalism and how it is performed, illustrating how Wikipedians consider the activities of others when considering a change as potential vandalism. The study is a case study considering when an individual should be banned from the site for contributing too much content that has been deemed vandalism. In the case of the decision to ban a vandal, the effort is to understand whether the activities are intentionally designed to corrupt content and wreak havoc on the community itself. This work illustrates that there is a fair amount of vandalism which is somewhat ‘routine’ but that some is still difficult to detect even by people who are practiced at looking for vandalism. Another interesting insight is that most tools that support vandalism rollback do not yet categorize or provide a prediction rating for the edit being viewed. Most tools simply show the edit, the prior version, and the IP or username of the editor who made it.

A second set of approaches rely on automated bots and algorithms. This approach has generated lots of research activity, so we focus on the prior work that is most related to how we have approached the problem.

Since 2007 automated bots have been widely used to fight vandalism in Wikipedia. The most prominent of them are ClueBot and VoABotII. Like many vandalism detection tools, they use lists of regular expressions and consult databases with blocked
users or IP addresses. The major drawback of these approaches is that most bots utilize static lists of obscenities and grammatical rules that are hard to maintain and, with some creativity, can be easily thwarted. A study on performance analysis of these bots in Wikipedia shows that they can detect about 30% of the instances of vandalism [105].

Several machine learning approaches have recently been proposed that would improve vandalism detection [105, 91, 24, 57]. Comparing the performance of these approaches is difficult because of several shortcomings in early vandalism detection corpora. These early corpora were too small, they failed to account for the true distribution of vandalism among all edits, and the hand labeling of the examples had not been double-checked by different annotators. These shortcomings were resolved by the creation of a large-scale corpus for the PAN 2010 competition consisting of a week’s worth of Wikipedia edits (PAN-WVC-10) [90].

The PAN 2010 corpus is comprised of 32,452 edits on 28,468 different articles. It was annotated by 753 annotators recruited from Amazon’s Mechanical Turk, who cast more than 190,000 votes. Each edit in the corpus was reviewed by a minimum of three annotators. The annotator agreement was analyzed in order to determine whether each edit is a regular edit or vandalism, with 2,391 edits deemed to be vandalism. The corpus is split into a training set and a test set, which have 15,000 and 18,000 edits, respectively [81].

A survey of detecting approaches [81] shows that about 50 features were used by the 12 different teams. Features are categorized into two broad groups: (1) edit textual features; (2) edit meta information features. Edit textual features are extracted based on the text of the edit. Some features in this category are adopted from previous work on spam detection in emails or blogs. For example, “Longest character sequence” and “upper case to low case char ratio” are known to be important features for spam...
detection in emails [50].

Traditional spam detection systems for emails mainly rely on textual features based on the content. Most features show existence of particular words or phrases [6, 70, 87, 103]. In some cases non–textual features are extracted from meta data. For example, whether or not a message contains attachments [6].

Edit meta information mainly contains two types of features: user features and comment features. In Wikipedia when an individual makes an edit, there is the opportunity to provide a short comment on the change. As a convention, many editors will use the comment to briefly explain the rationale for the change or what type of change is being made. Comment features are extracted based on the comment related to an edit. Most teams used comment features, but two teams extensively relied on user features. User features are extracted based on the editing pattern of the user.

Identifying an editing pattern requires that some amount of state, or history, be maintained. Some forms of UGC do not maintain history as a function of the system design, but in the case of wikis user history is a given. Two teams in the PAN competition relied on user reputation features extracted from the edit revision history. Those teams placed second and third in the competition. Other approaches rely more heavily on a model of user reputation. Adler et.al. [2] used WikiTrust to estimate user reputation. In their system, users gain reputation when their edits are preserved and they lose reputation when their edits are reverted or undone [4].

Javanmardi et.al. [61] used user reputation features based on the reputation management system they developed earlier. Compared to [4], the model is simpler and more efficient. One reason is that it is only based on the stability of inserts. In [4], stability of deletes and reverts are also considered. A detailed comparison between these two approaches are presented in [61].
Potthast et al. [81] combined the predictions submitted by the top 8 teams and developed a meta classifier based on the predictions. To learn the meta classifier, they used random forest. This improved the classification performance (ROC–AUC) of the top team significantly, from 0.91580 to 0.9569. Using a similar approach, Adler et al. [107] developed a meta classifier based on the predictions of three different classifiers. To evaluate the meta classifier, they merged train set and test set and reported ROC–AUC based on 10–fold cross validation. Hence, their results are not comparable with PAN competition results and our results.

In general, meta classifiers work at a macro level by aggregating results from different classifiers. However, this makes them even more complex and less practical for real–time applications. In contrast, in this study we work at the level of individual features and focus on building accurate classifiers with a minimum set of features. We show that the classification performance of the compact classifier is comparable to the meta classifier developed in [81].

4.4 Feature Extraction

To extract features, we mine the entire English Wikipedia history dump, released on Jan, 2010. Totally, 41 edits in PAN corpus are missing in the dump. We use crawler4j to extract the data of these missing edits. We also use other Wikipedia SQL dumps to extract users with special access rights such as administrators and bureaucrats.

Using all this data, we extract 66 features. This feature set includes most of the features used in the PAN competition by different teams [81]. In addition, we introduce some new features. Table 4.1 shows the features along with their definitions (each row may represent more than one feature).

\[^{1}\text{http://code.google.com/p/crawler4j/}\]
Similar to [81] we separate edit textual features and edit meta data features. Since the edit meta data features contains both user and comment features, we consider them as different groups. In addition, we add language model features as a new group. These features capture topical relevance and are estimated based on the language model of the newly submitted revision and the background. The effectiveness of using these features for vandalism detection has been studied in [24, 85]. We categorize the features into four groups (see Table 4.1):

- **User Features**: In this work we introduce 12 features for each user including statistical and aggregate features. We calculate these features by mining history revisions up to time $T$ [61]. For the purpose of this study, we consider $T$ as 2009–11–18 which is the timestamp of the earliest edit log in the PAN corpus. Hence, all features in this category are based solely on history data.

- **Textual Features**: we have 30 features in this category. Most of the features in the category are adopted from [112]. We calculate the value of the features based on the inserted content. In addition, in this work, we calculate the value of the features also based on the deleted content. To distinguish these features we use “Ins” and “Del” prefix throughout this paper. For example, Vulgarism shows the frequency of vulgar words. We expect insertion of vulgar words to be a signal for vandalism. Conversely, we expect deletion of such words to be a signal for legitimate edits aiming at removing vandalism.

- **Meta Data Features**: we have 22 features in this category. Most features are extracted from the comments associated with the edits. For example, we have similar textual features to the previous category but here we extract them based on the comment. Because the descriptions are similar to their peer textual features, we do not define them in Table 4.1. These features are specified by *.
In addition to these, we introduce some new features that we extract from the automatic comments generated by Wikipedia. These comments specify which section of the article has been edited. We extract unigrams, bigrams, and trigrams from these types of comments. We use feature selection on the PAN train set to extract the important ones. For example, the short time interval between the old and the new revisions might be an indicator of vandalism.

- **Language Model Features:**
  
  In this category we have 3 features which calculate the Kullback–Leibler distance (KLD) between two unigram language models. We calculate KLD between the previous and the new revision [24]. We introduce two more features: the KLD between the inserted content and the previous revision. Similarly, we calculate the KLD between the deleted content and the previous revision. We suspect that, sometimes vandalism comes with some unexpected words so we expect to see sharp changes in the distance. Conversely, deleting unexpected words can be an indicator of legitimate edits.

### 4.5 Learning Vandalism Detection Model

We consider vandalism detection as a binary classification problem. We map each edit in PAN corpus into a feature vector and learn the labels by mapping feature vectors onto \{0,1\}, where 0 denotes legitimate edit, and 1 vandalistic edit. To learn a classifier and tune its free parameters, we use PAN train set and keep the test set untouched for final evaluation.

We use different binary classification algorithms which have been widely applied to spam detection such as Naive Bayes [70], Logistic Regression [17], and SVM [103]. We
Table 4.1: List of features. Asterisked features are extracted based on both the edit content and its comment.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSR, DDSR, Rep</td>
<td>Aggregated features representing a user’s reputation</td>
</tr>
<tr>
<td>Ins Words</td>
<td>Total number of words inserted by a user</td>
</tr>
<tr>
<td>Del Words</td>
<td>Total number of words deleted by a user</td>
</tr>
<tr>
<td>Lost Words</td>
<td>Total number of deleted words from a user</td>
</tr>
<tr>
<td>Ins Revision</td>
<td>Total number of revisions a user has done insertion in</td>
</tr>
<tr>
<td>Del Revision</td>
<td>Total number of revisions a user has done deletion in</td>
</tr>
<tr>
<td>Ins Page</td>
<td>Total number of pages a user has done insertion in</td>
</tr>
<tr>
<td>Del Page</td>
<td>Total number of pages a user has done deletion in</td>
</tr>
<tr>
<td>User Type</td>
<td>User has some special rights, such an admin, a bot, or a bureaucrat</td>
</tr>
<tr>
<td>User Page</td>
<td>User has a user page in Wikipedia</td>
</tr>
<tr>
<td>Ins Size</td>
<td>Number of inserted words</td>
</tr>
<tr>
<td>Del Size</td>
<td>Number of deleted words</td>
</tr>
<tr>
<td>Revision Size</td>
<td>Size difference ratio between the old and the new revision.</td>
</tr>
<tr>
<td>Blanking</td>
<td>The whole article has been deleted</td>
</tr>
<tr>
<td>Internal Links</td>
<td>Number of links added to Wikipedia articles</td>
</tr>
<tr>
<td>External Links</td>
<td>Number of added external links</td>
</tr>
<tr>
<td>Word Repetitions</td>
<td>Length of the longest word</td>
</tr>
<tr>
<td>Char Repetitions</td>
<td>Length of the longest repeated char sequence</td>
</tr>
<tr>
<td>Compressibility</td>
<td>Compression rate of the edit differences.</td>
</tr>
<tr>
<td>Capitalization*</td>
<td>Ratio of upper case chars to lower case chars</td>
</tr>
<tr>
<td>Capitalization All*</td>
<td>Ratio of upper case chars to all chars</td>
</tr>
<tr>
<td>Digits*</td>
<td>Ratio of digits to all letters</td>
</tr>
<tr>
<td>Special Chars*</td>
<td>Ratio of non-alphanumeric chars to all chars</td>
</tr>
<tr>
<td>Diversity*</td>
<td>Length of all inserted lines to the (1 / number of different chars)</td>
</tr>
<tr>
<td>Inserted Words*</td>
<td>Average term frequency of inserted words</td>
</tr>
<tr>
<td>Vulgarism*</td>
<td>Frequency of vulgar words</td>
</tr>
<tr>
<td>Bias*</td>
<td>Frequency (impact) of biased words</td>
</tr>
<tr>
<td>Sex*</td>
<td>Frequency (impact) of sex related words</td>
</tr>
<tr>
<td>Spam*</td>
<td>Frequency (impact) of spam related words</td>
</tr>
<tr>
<td>Pronouns*</td>
<td>Frequency (impact) of personal pronouns</td>
</tr>
<tr>
<td>WP*</td>
<td>Frequency (impact) of mark up related words</td>
</tr>
<tr>
<td>Special Words*</td>
<td>Aggregation of vulgarism, bias, sex, spam, pronouns, and WP ratios</td>
</tr>
<tr>
<td>Time Diff</td>
<td>Time interval between submission of the old and new revision</td>
</tr>
<tr>
<td>Category</td>
<td>If the automatic comment contains “category”</td>
</tr>
<tr>
<td>Early Years</td>
<td>If the automatic comment contains “early years”</td>
</tr>
<tr>
<td>Copyedit</td>
<td>If the automatic comment contains “copyedit”</td>
</tr>
<tr>
<td>Personal Life</td>
<td>If the automatic comment contains “personal life”</td>
</tr>
<tr>
<td>Revert</td>
<td>If the automatic comment contains “revert”</td>
</tr>
<tr>
<td>Revision Ordinal</td>
<td>Ordinal of the submitted revision</td>
</tr>
<tr>
<td>Length</td>
<td>Length of the comment</td>
</tr>
<tr>
<td>Reverted</td>
<td>if the MD5 digest of new revisions is the same as one of the old ones in window size of 10</td>
</tr>
<tr>
<td>KL Distance</td>
<td>KL distance between the old revision and the new revision</td>
</tr>
<tr>
<td>KL Distance Ins</td>
<td>KL distance between the inserted words and the new revision</td>
</tr>
<tr>
<td>KL Distance Del</td>
<td>KL distance between the deleted words and the new revision</td>
</tr>
</tbody>
</table>
Table 4.2: Classification performance for the binary classifiers.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AUC (CV)</th>
<th>AUC (Test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.9482 ± 0.0057</td>
<td>0.9068</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.9494 ± 0.0070</td>
<td>0.9128</td>
</tr>
<tr>
<td>SVM (LibSVM, RBF Kernel, Gamma = 0.15, c=11)</td>
<td>0.9537 ± 0.0007</td>
<td>0.9202</td>
</tr>
<tr>
<td>Random Forest (1000 trees)</td>
<td>0.9739 ± 0.0024</td>
<td>0.9553</td>
</tr>
</tbody>
</table>

also use random forests which has been shown to be a very effective binary classifier [16]. Table 4.2 shows the classification performance for 3–fold cross validation on the train set and also on a test set. We use the Java implementation of these binary classifiers from Weka [46] and tune the free parameters based on cross validation. Because we use the train set to learn some of the features, and to make our results comparable to the PAN results, we do not use the test set in any way during training. The test set is only used to measure classification performance. Here we report classification performance in terms of area under the ROC curve (ROC–AUC). (The ROC-AUC metric was used in the PAN 2010 evaluation. Results for area under the precision-recall curve are similar.) In all our experiments random forests outperformed the other classifiers.

4.5.1 Vandalism Detection Using Random Forest

Random forest is not widely used for spam detection because it is known to be expensive in terms of time and space [108]. In this section we explain how we can train random forest classifiers efficiently and gain high classification performance at the same time.

Statistical analysis of Wikipedia edits show that roughly 7% of edits are vandalistic.
which is consistent with the vandalism ratio in PAN corpus. Given this, we need to use machine learning algorithms which are robust to imbalanced data, such as random forests. In addition, random forests are a suitable option for datasets with missing data [15]. The PAN data set is an imbalanced dataset and some features such as the user group features are sometimes missing; For 4% of users we do not have any user group information at all. This suggests that a random forest model could have benefits over other techniques that may not deal as well with imbalanced data or missing features. One advantage of the ROC-AUC metric is that it is fairly insensitive to imbalanced data.

To learn a random forest classifier, we need to tune two free parameters: the number of trees in the model and the number of features considered to split each node. Our experiments show that classification performance is sensitive to the former but not to the latter. This result is consistent with Breiman’s observation [15] on the insensitivity of random forests to the number of features considered in each split.

To tune the number of trees, we partition the train set into three folds and use 3–fold cross validation. Using three folds allowed us to keep a reasonably large number of vandalized cases in each training set (around 600). To find the optimal value for the number of trees, we need to sweep a large range of values. Hence, we need to design an efficient process for this purpose.

For each fold, we create a pool of $N = 10,000$ trees, each trained on a random sample of the training data in that fold. Then we use this pool for creating random forests of different sizes. For example, to create a random forest with 20 trees, we randomly select 20 trees from this pool of $N$ trees. However, since this random selection can be done in $C(N, 20)$ different ways, each combination may result in a different AUC. We repeat the random selection of trees $r = 50$ times and we report the mean and variance of the $F \times r$ results (where $F$ is the number of folds).
The advantage of this approach is that we can calculate the mean and variance of AUC very efficiently for forests with different sizes without the need to train a huge number of trees. Otherwise, to report the mean and variance of AUC for random forests of size \( k = 1 \) to \( T \), we would need to train \( r + 2 \times r + 3 \times r + \ldots + T \times r = r \times T(T + 1)/2 \) trees for each fold, which is \( 10^8 \) trees. Using our approach we only need to train \( N \) trees per fold (in our experiments we used \( N = 5 \times T \)).

Figure 4.1 shows the mean of AUC as a function of number of trees in the model. As more trees are added to the model, mean of AUC increases and the variance decreases. The mean of AUC does not improve significantly after more than about 500 trees in the random forest but the variance continues decreasing. It should be emphasized that models with smaller variance are more stable and therefore more predictable in test environments. Although more trees may result in slightly better AUC values, we decide to set the number of trees at 1000 to have a balance between classification performance and model complexity. More complex models with more trees would require more time for prediction in a real–time application. Given this, the AUC on for 3–fold cross validation on the train set is \( 0.9739 \pm 0.0024 \). The AUC value on PAN test set is 0.9553. This result is significantly higher than the best AUC reported to the PAN competition which was 0.9218 [81].

### 4.6 Feature Selection

The classifier mentioned in the previous section makes its decision based on 66 features which fall into four logically different groups. However, computing and updating each of these features imposes significant off–line cost at run–time. For example, computing and updating features in the user group requires the tracking of all edits done by each individual. Maintaining a system that updates data for computing
these features would come at a cost for the wiki. Some other features like textual features are computed after submission of a new edit and the vandalism detection system should be able to compute them in real-time.

In this section, we report the results of our experiments in finding a minimum set of features whose classification performance is almost as good as the one with all 66 features. In other words, we try to detect and eliminate redundant or unnecessary features. We consider two types of redundant or unnecessary features: (a) features that are not informative and do not help in discriminating legitimate and vandalistic content; (b) features correlated with some other features so that once one of them is selected, adding others does not add any new discriminating power to the model.
In order to detect and eliminate redundant features, we perform two sets of experiments. First, in Section 4.6.1 we study the contribution of groups of features as a whole unit to examine if any of the four groups can be eliminated without a significant drop in AUC. Then in Section 4.6.2 we study the contribution of each feature individually and use the results of this analysis to eliminate redundant features.

Given the large number of experiments needed for this study, we use the Amazon MapReduce cluster to run them in parallel. In our implementation, each mapper receives specific config information and trains a classifier for that configuration. Then reducers aggregate the AUC results for different folds and report the mean AUC for the different configs.

### 4.6.1 Eliminating Groups of Features

For each group of features, we train a classifier without the features in that group. This will show us the drop in AUC when this group is ignored. Table 4.3 shows the results. These results indicate that removing features in user group and textual group results in large drops in AUC, while the drop in AUC for the meta data and the language model groups is much less.

Based on these results we can not infer that features in the Meta data and LM group are not informative. The only conclusion is that once we have both user and textual features in our feature set, adding meta data and language model features does not add substantial additional discriminating power to the model. Table 4.4 supports this interpretation: when we only use meta data or language model features the AUC value is much higher that of a random classifier (0.50). The single group of features that results in the highest AUC (0.9399) is the user group.
Table 4.3: The drop in mean AUC when we eliminate a feature group. Results on train set are for 3-fold cross validation.

<table>
<thead>
<tr>
<th>Dropped Group</th>
<th>Train set</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>−3.8%</td>
<td>−6.6%</td>
</tr>
<tr>
<td>Textual</td>
<td>−1.8%</td>
<td>−1.5%</td>
</tr>
<tr>
<td>Meta data</td>
<td>−0.4%</td>
<td>−0.3%</td>
</tr>
<tr>
<td>Language Model</td>
<td>−0.2%</td>
<td>−0.3%</td>
</tr>
</tbody>
</table>

Table 4.4: The mean AUC when we only use features in one group.

<table>
<thead>
<tr>
<th>Selected Group</th>
<th>Train set</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>0.9399</td>
<td>0.9225</td>
</tr>
<tr>
<td>Textual</td>
<td>0.9001</td>
<td>0.8400</td>
</tr>
<tr>
<td>Meta data</td>
<td>0.7019</td>
<td>0.6924</td>
</tr>
<tr>
<td>LM</td>
<td>0.7271</td>
<td>0.6986</td>
</tr>
<tr>
<td>Baseline (all groups)</td>
<td>0.9739</td>
<td>0.9553</td>
</tr>
</tbody>
</table>

The results of this study might not seem consistent with what was reported in [107]. Here we show the importance of user features, while in [107], authors have concluded that there is much less need to care about the past performance of the users and therefore that user features do not contribute in any significant way to the classification performance. We see two reasons for this difference. First, we calculate user features based on different approaches; so our feature sets are not the same. Secondly, the authors in [107] have used meta classifiers to measure the contributions of groups of features to the classification performance.

It is important to note that a meta classifier works at the macro level. In other words, it sends the same test instance to a set of classifiers and then aggregates their decisions (e.g., using a weighted average). In many problems this approach produces more accurate results [16] because now the final decision is made based on the votes casted by a diverse set of classifiers. However, training different classifiers on different groups of features and then combining them with a meta classifier, as suggested in [107], may not necessarily result in an optimal classification performance.
The reason for this is that, by limiting each single classifier to a subset of features, we may limit its expressiveness, thereby making it weaker. A combination of such classifiers may only result in a sub-optimal performance; but a classifier that has access to all of the features in different groups, has the chance of making its decision based on a combination of features in different groups. For example, a decision tree based classifier may grow branches in which both user reputation features and textual features are used. Therefore, it can exploit all of the useful information embedded in these features to enhance the result.

4.6.2 Eliminating Individual Features

In Section 4.6.1 we showed that all the four groups contain informative features. In this section, we attempt to find the smallest feature set whose AUC is comparable to the AUC of a classifier with 66 features. To this aim we do feature selection.

There are three different approaches for performing feature selection, univariate feature analysis, wrapper-based methods, and proxy methods [45, 67]. Univariate feature analysis methods such as Information Gain or Chi-Square evaluate features independently in order to estimate their importance. Because these methods evaluate the utility of each feature in isolation, they are unable to detect and benefit from correlations among the features and often yield unnecessarily large feature sets.

In wrapper-based methods one wraps a feature selection process such as forward stepwise selection or backwards stepwise elimination around the learning algorithm that will be used to train the final model so that the feature selection process can find a small set of features that works well with that learning method. Because the contribution of each feature is evaluated in the context of the other features used in the model, wrapper-based methods are very effective at eliminating features whose
contribution to the model is redundant with other features already in the model, and thus often yield the smallest feature sets. The main difficulty with the wrapper approach is that it can be prohibitively expensive to wrap feature selection around expensive learning methods such as random forests. For this reason, proxy feature selection methods are often used. In proxy methods, feature selection is performed for a simpler, more computationally tractable model class such as linear or logistic regression models, and the features found to be most important for the simpler model class is then used in the more complex, more expensive model class (in this work random forests). Because proxy methods do not take the specific learning algorithm into account, they do not always find as compact a set of features as wrapper methods. However, in practice there usually is strong overlap between the features that are best for a simpler model and those that are effective in more complex models, so proxy methods often yield feature sets that are almost as small as those returned by wrapper methods.

Because the model class that appears to perform best on vandalism detection is computationally expensive (random forests), in this work we consider only proxy feature selection methods. We need a feature selection algorithm that is efficient and which considers correlations between features and is able to detect and eliminate redundant features effectively. We use Lasso Logistic Regression (Least Absolute Shrinkage and Selection Operator for Logistic Regression) [34] for this purpose. This fits a regularized logistic regression model to features in such a way that the final model has a sparse solution in the feature space. Thus, the weight of redundant features in the final model would be zero. It means that we can remove these features from the model with no significant change in the classification performance. For this, we use the Logistic Regression Lasso implemented in glmnet package in R [34].

Lasso for logistic regression has a regularization parameter, $\lambda$, that is a trade off
between sparsity of the model and its classification performance. Lower values for $\lambda$ result in more relaxation of the regularization constraint which allows more features to have non-zero weights. The R package, glmnet [34], uses regularized maximum (binomial) likelihood to fit this regularized logistic regression model to the data. The problem can be formalized by the following:

$$\max_{(\beta_0, \beta) \in \mathbb{R}^{p+1}} \left[ l(\beta_0, \beta) - \lambda \| \beta \|_1 \right]$$ (4.1)

where

$$l(\beta_0, \beta) = \frac{1}{N} \sum_{i=1}^{N} y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i))$$ (4.2)

and

$$p(x_i) = \frac{1}{1 + \exp(\beta_0 + x_i^T \beta)}$$ (4.3)

Table 4.5 shows the features selected for different values of $\lambda$. For $\lambda = 0.0716$ only one feature is selected. This means that according to lasso if we want to make the classification model based on only one feature, “Ins Special Words” would be our best choice. As we decrease the value of $\lambda$, more features are selected according to their importance. The second most important feature is “DDSR”. The last column in Table 4.5 shows the value of the AUC on the PAN test set for classifiers trained on the
Table 4.5: Feature selection using lasso. Parameter $\lambda$ determines a trade off between number of selected features and AUC of the classifier. Smaller values of $\lambda$ allow more features be selected and result in models with higher performance.

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Selected Features</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0716</td>
<td>Ins Special Words</td>
<td>0.5974</td>
</tr>
<tr>
<td>0.0594</td>
<td>DDSR, Ins Special Words</td>
<td>0.8614</td>
</tr>
<tr>
<td>0.0375</td>
<td>DDSR, Rep, Ins Special Words</td>
<td>0.8965</td>
</tr>
<tr>
<td>0.0340</td>
<td>DDSR, Rep, User Page, Ins Digits, Ins Special Words</td>
<td>0.9074</td>
</tr>
<tr>
<td>0.0310</td>
<td>DDSR, Rep, User Page, Ins Digits, Ins Vulgarism, Ins Special Words</td>
<td>0.9090</td>
</tr>
<tr>
<td>0.0257</td>
<td>DDSR, User Page, KLDNew2OLD, Ins Digits, Ins Vulgarism, Ins Special Words</td>
<td>0.9197</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0.0030</td>
<td>DDSR, Del Words, User Type, User Page, Copyedit, Personal Life, Revision Ordinal, Comment Length, KLDNew2OLD, Blanking, Ins Internal Link, Ins External Link, Longest Ins Word, Ins Longest Char Seq, Ins Compressibility, Ins Capitalization, Ins Digits, Ins Special Chars, Ins Vulgarism, Ins Bias, Ins Sex, Ins Pronouns, Ins WP, Ins Special Words, Del Bias, Ins Digits, Comment Special Chars, Comment Spam</td>
<td>0.9505</td>
</tr>
</tbody>
</table>

selected features. As the number of selected features increases, we see a higher value for AUC but the cost of computing and updating the features would also increase.$^2$

We use 3–fold cross validation on the PAN train set to pick the largest value for $\lambda$, where the drop in AUC is not statistically significant. The result was $\lambda = 0.0030$ which leads to selection of 28 features. Table 4.5 shows a list of the selected features. The AUC for this feature set on PAN test set is 0.9505. This feature set only includes less than half of the original features but the drop in AUC is only 0.005.

In this sparse feature set we have features from all groups. For example, “DDSR”, “Del Words”, “User Type”, and “User Page” are selected from the user group. If we follow the Lasso path, features get added and eliminated as $\lambda$ decreases. For example,$^2$

---

$^2$The lasso method does not necessarily yield a monotonically increasing set of features; it is possible that as $\lambda$ is decreased some features that were in the set for larger $\lambda$s might be removed from the set as other feature replace them.
“Rep” is selected as the third important feature, but because of its correlation with other selected features it is eliminated from the feature set later. Interestingly, “Rep” is computationally more expensive than “DDSR”.

We have 17 features from the textual group. These features are also widely used for spam detection in other domains such as emails or blogs [50]. For example, “Ins Longest Char Repetitions” shows whether a user has inserted a long sequence of the same character which can be a good indicator of vandalism. There are also some textual features which are unique to Wikipedia. For example, “Del Bias” shows that the user has deleted words which represent bias and therefore is a good indicator of legitimate edit.

We have 6 features selected from the meta data group. For example, “Comment Length” or “Comment Special Chars” are selected as important features. “Personal Life” is another important feature. It shows whether the edit is made in the “Personal Life” section of a biography article. We have observed that this section of biography articles is more often vandalized and therefore this feature can be an important signal. This observation is consistent with Wikipedia statistics which shows high vandalism ratio in biography articles\(^3\). Given that Wikipedia automatically adds the name of an edited section to the comment associated to each edit, we can extract this feature from comments.

The only feature that is selected from the Language Model group is “KLDNew2OLD”. The goal of this feature is to detect sharp linguistic distance between the new and the previous revision which can be an important signal for vandalism detection.

\(^3\)http://en.wikipedia.org/wiki/Wikipedia\ biography\ controversy
4.7 Cost Sensitive Feature Selection

In Section 4.6 we focused on features selection in order to find a small subset of available features that would encapsulate all useful information for the task at hand. This was useful to eliminate redundant features and increase the interpretability of the selected features. However, in many applications, it is also useful to take into account the amount of effort required to compute the features. The importance of feature cost is ignored by many machine learning methods [33].

The general setting of cost sensitive learning is consists of designing classification models while taking into account the cost involved in the entire decision process. This includes the cost of data acquisition, the cost of labeling the training samples and the cost of making different decision errors [102]. In this work we focus on the first case where there are different costs to acquire different features. For example, to calculate value of the feature “Rep” we need to process all history revisions and track contributions of each individual user; we also need to track how other users have edited their contributions [61]. Some other features like “Comment Length” are easier to compute and only need a little light text processing.

To address this issue here we try to take into account the cost of acquiring different groups of features during the learning process. We are interested in cases where well–performing cheap features are preferred over expensive ones with slight difference in overall classification results. For this purpose, we study two different scenarios. In the first scenario features are selected based on their corresponding costs.
4.7.1 Cost Sensitive Lasso

In this section we use regularized logistic regression based on lasso to implement cost sensitive feature selection. Unlike in 4.6.2 where we considered a fixed penalty $\lambda$ for all features, here we assign higher penalties to the computationally more expensive features. We use glmnet package in R [34] which allows different penalties $\lambda_j$ for each of the variables via a penalty scaling parameter $\gamma_j \geq 0$. If $\gamma_j > 0$, then the penalty applied to $\beta_j$ is $\lambda_j = \lambda \gamma_j$. If $\gamma_j = 0$, that variable does not get penalized, and enters the model unrestricted at the first step and remains in the model. Here we set the values of each $\gamma_j$ relative to the cost of acquisition of the feature; in the previous scenario $\gamma_j$ was set to 1 for all the features.

Here the costs assigned to features vary between 1 and 5; 1 is for the least computationally expensive feature and 5 is for the most computationally expensive one. Figure 4.2 shows the histogram of cost values in our feature set along with some examples in each category.

Considering these features’ costs we run lasso for logistic regression. Table 4.6 shows how features are selected in the lasso path along with the value of AUC. “User Page” whose cost is 1 is the first feature that is selected. As features are added to the feature set AUC tends to increase. However, there is a significant jump in AUC when “DDSR” is added to the feature set, from 0.8245 to 0.9065. This feature is among the most computationally expensive features but because of its importance it has been selected in the first steps of feature selection. This result is consistent with the results in Table 4.5 where “DDSR” was selected as the second most important feature. In that case we assumed similar cost for all the features while here we assigned high cost to this feature. The fact that this feature is selected in early in the cost-sensitive feature selection process despite it’s high cost, and the large gain in AUC, suggests
the importance of this user feature.

After selecting 33 features, AUC is comparable to the AUC of a classifier with 66 features. This means that having about half of the features in the feature set the classification performance is roughly the same. However, in contrast to Section 4.6.2, here we are training classifiers using a similar number of features but they are less computationally expensive features.

### 4.7.2 Cost Sensitive Group Lasso

In this section we do cost sensitive feature selection but also take into account the data sources the features are extracted from. We categorize the features into five
Table 4.6: Cost Sensitive Feature Selection using Lasso. The lasso path shows how
features are added incrementally and the changes in AUC on test set.

<table>
<thead>
<tr>
<th>λ</th>
<th>Feature Set Size</th>
<th>Cost</th>
<th>Newly Selected Features</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0882</td>
<td>1</td>
<td>1</td>
<td>User Page</td>
<td>0.7129</td>
</tr>
<tr>
<td>0.0882</td>
<td>2</td>
<td></td>
<td>Ins Special Words</td>
<td>0.7581</td>
</tr>
<tr>
<td>0.0476</td>
<td>3</td>
<td>1</td>
<td>Revision Ordinal</td>
<td>0.6832</td>
</tr>
<tr>
<td>0.0447</td>
<td>4</td>
<td>2</td>
<td>Ins Character Repetition</td>
<td>0.7087</td>
</tr>
<tr>
<td>0.0396</td>
<td>5</td>
<td>1</td>
<td>User Type</td>
<td>0.7266</td>
</tr>
<tr>
<td>0.0291</td>
<td>6</td>
<td>2</td>
<td>Ins Vulgarism</td>
<td>0.7235</td>
</tr>
<tr>
<td>0.282</td>
<td>7</td>
<td>2</td>
<td>Ins Digits</td>
<td>0.8245</td>
</tr>
<tr>
<td>0.0265</td>
<td>8</td>
<td>4</td>
<td>DDSR</td>
<td>0.9065</td>
</tr>
<tr>
<td>0.0257</td>
<td>9</td>
<td>2</td>
<td>KL Distance</td>
<td>0.9299</td>
</tr>
<tr>
<td>0.242</td>
<td>10</td>
<td>2</td>
<td>Ins Bias</td>
<td>0.9305</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0021</td>
<td>32</td>
<td>2</td>
<td>Del Special Words</td>
<td>0.9486</td>
</tr>
<tr>
<td>0.0019</td>
<td>33</td>
<td>3</td>
<td>Ins Revision</td>
<td>0.9516</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0014</td>
<td>58</td>
<td>4</td>
<td>Del Words</td>
<td>0.9543</td>
</tr>
<tr>
<td>0.0001</td>
<td>59</td>
<td>2</td>
<td>KL Distance Ins</td>
<td>0.9553</td>
</tr>
</tbody>
</table>

groups; where features in one group are captured in the same way and based on the
same data source. For example, when we process text of history revisions and extract
word ownerships, we have the data to calculate features like “DSR”, “DDSR”, and
“Rep”. In other words, when we pay the cost of processing text of history revisions to
calculate a feature like “Rep” then the cost of calculating a new feature like “DSR”
will be zero. Group Lasso [84] captures this notion and provides a way for cost
sensitive feature selection. When a feature from a group is selected, other features of
that group will be selected automatically.

We acquire features in five different ways: (1) processing text of all history revisions of
articles; (2) processing the text of current revision of articles; (3) processing meta data
of current revision of articles; (4) processing sql dumps; and (5) crawling Wikipedia
user pages. We process the text of history revisions to track contributions of users
in order to acquire some features like “DSR”, “DDSR”, and “Rep”. For some other
features like “Time Diff”, “Category”, or “Comment Length” we do not need to process the entire history dump and processing the meta data of the current revisions is enough. We use Wikipedia sql dumps to acquire features like “User Type”. We also crawl Wikipedia user pages in order to calculate the value of features like “User Page”.

To be able to incorporate group information into our feature selection process, we use group lasso as implemented in the package ”grplasso” [84] in R. ”grplasso” estimates the weights of a logistic function, considering the group information for different features. Logistic group lasso estimator $\beta$ can be computed by the maximizing the following convex function:

$$\max_{(\beta_0, \beta) \in \mathbb{R}^{p+1}} \left[ l(\beta_0, \beta) - \lambda \sum_{g=1}^{G} s(df_g) \| \beta_g \|_2 \right]$$ (4.4)

Here, the loss function $l$ is defined as in equation 4.2 and $\beta_g$ is the parameter vector corresponding to the $g$th group of the predictors. The function $s(.)$ is used to rescale the penalty with respect to the dimensionality of the parameter vector $\beta_g$, the default value of which is set to $s(df_g) = df_g^{\frac{1}{2}}$ to ensure that the penalty term is of the order of the number of parameters $df_g$.

Table 4.7 shows the result. The first group selected is Group 5 which has one feature “User Page”. Then Group 4 is added which has one feature, “User Type”. Group 1 is added in the third step which has 10 features. All these three groups contain user related features. When Group 3 is added we see a significant jump in the value of AUC, about 16%. It shows the importance of user group features.

The next group which is selected is group 2 which mostly consists of textual features. Adding this group, we see some improvements in the AUC. The last selected group is
<table>
<thead>
<tr>
<th>( \lambda )</th>
<th>Feature Set Size</th>
<th>Selected Group</th>
<th>( \text{AUC} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>350</td>
<td>1</td>
<td>Group 5</td>
<td>0.7129</td>
</tr>
<tr>
<td>330</td>
<td>2</td>
<td>Group 5+4</td>
<td>0.7581</td>
</tr>
<tr>
<td>300</td>
<td>13</td>
<td>Group 5+4+1</td>
<td>0.9230</td>
</tr>
<tr>
<td>50</td>
<td>45</td>
<td>Group 5+4+1+2</td>
<td>0.9519</td>
</tr>
<tr>
<td>10</td>
<td>66</td>
<td>Group 5+4+1+2+3</td>
<td>0.9555</td>
</tr>
</tbody>
</table>

Group 3 which contains meta data features. The small increase in the value of AUC shows that adding this group of features does not improve AUC significantly.

According to group lasso, user features are the most important features, confirming the results of feature selection experiments from the previous sections. Although these features are computationally expensive to acquire compared to other features, they are selected in early steps of the lasso path because of their significant contribution to classification performance. This is very interesting because in traditional spam detection systems for web content or emails there are generally no user related features. Maintaining a history of user activity in UGC systems is becoming more common. Many wikis maintain history which makes computing user features possible. But that still opens the question of how to maintain user history in a way that computing user level features is fast and incremental.

### 4.8 Conclusion

In this chapter we described a machine learning approach to detect vandalism in User Generated Content (UGC) systems. Our specific example comes from Wikipedia, but the approach is general and can be applied to many forms of UGC.

Based on the validated corpus of Wikipedia edits from the PAN competition we
trained and tested several binary classifiers using learning methods that have been widely used for spam detection: Naive Bayes, Logistic Regression, and SVMs. We also trained models using random forests which have not been widely used for spam detection. Interestingly the results show that the random forests significantly outperform the other three types of classifiers on this problem. An additional benefit of random forests is that they are robust to missing and unbalanced data which is a common characteristic of vandalism/spam data sets.

The common practice when training models for spam detection has been to train the models using all available features. Because our goal is to develop models that are small and fast enough to be used in a real–time vandalism detection tool, we used feature selection to eliminate redundant features so that the computational complexity of the final model is as small as possible while retaining high accuracy. Some of the features that proved to be most informative require mining user action histories. Computing the value of these features can be expensive. Because of this, we considered both traditional feature selection, as well as cost sensitive feature selection that takes into account the cost of acquisition of each feature. We compressed the learned model by estimating the contribution of different features and feature groups to the random forest model. Across the four groups of features we found that each group contained some important features, but the user features representing the history of user contributions are most important, so we could not ignore this group of features. This motivated a focus on individual features to determine which specific features (instead of groups of features) contributed most significantly to the model. Using lasso, we found a minimum set of features whose classification performance is almost as good as the one with all 66 features. Furthermore, we did cost sensitive feature selection to force learning to prefer well–performing cheap features over more expensive features that yield only slight advantage in classification results. In combination, these techniques help us train a compact, sparse vandalism detection model.
that scales well and executes in real-time.
Chapter 5

Handling Class Imbalance Problem

5.1 Summary

A classifier built from a data set with a highly-skewed class distribution tend to predict the more frequently occurring classes more often than the infrequently occurring classes. This is largely due to the fact that most classifiers are designed to maximize accuracy. In many applications, this classification behavior is unacceptable because the minority class is the class of primary interest (i.e., it has a much higher misclassification cost than the majority class). In this chapter we study the problem in the context of vandalism detection. Using random forests, we try different resampling and cost-sensitive learning methods. Our experiments show that these methods do not improve the classification performance significantly for forests with large number of trees.
5.2 Introduction

Traditional machine learning classifiers aim at improving accuracy – the number of cases correctly classified. Traditional classifiers are not optimal on data sets that contain orders of magnitudes more negatives than positives, as those classifiers are trained mostly on negative data cases and get biased towards negatives\(^1\). As an example, in a data set where 99% are negative data cases, a naive algorithm always predicting negative will classify correctly 99% of the time. This algorithm can not be considered an acceptable classifier because predicting the 1% positives may be crucial.

In many real–life applications such as cancer–detection, credit–card fraud–prevention, or fault–detection in nuclear power plants: positive cases happen more rarely than negatives. In some cases, minority cases are actually what we want to detect and know about. Therefore, accuracy, defined as “the rate of correct predictions made by the model over a data set” [94], is not a relevant measure of a classifier’s effectiveness.

There are several methods that can be of use when dealing with skewed class distributions with unequal misclassification costs. The methods we analyze in this chapter all can be considered forms of resampling or cost–sensitive learning. We apply these methods on PAN 2010 data set [81]. The goal is to see if any of these methods can improve the classification performance of random forests.

This chapter is structured in the following way. We begin with a review of the relevant work on class imbalance problem. Through this we identify a number of commonly used approaches to handle the class imbalance problem. In subsequent sections we apply resampling and cost sensitive learning methods. We conclude by discussing the

\(^1\)Holding with the established practice, the minority class is designated the positive class and the majority class is designated the negative class.
5.3 Background

In our data set, positives constitute roughly 7%, which equates to an imbalance rate of 1:14 (i.e. there are 14 times more negatives than positives). While this imbalance rate is only moderate (one order of magnitude from perfect balance), we assume that some techniques could be used to combat imbalance when learning on the PAN data set. Traditionally, there are three main approaches to handle imbalance: fixing the imbalance, circumventing the imbalance, adapting existing algorithms [118]. In this work we focus on approaches geared towards fixing the imbalance, in particular oversampling, undersampling, and cost–sensitive learning methods. These methods can be applied before and independently of any classifier. However, the effectiveness of a technique may vary among different classifiers.

5.3.1 Resampling techniques

Resampling techniques aim at addressing at least two problems that frequently happen in imbalanced data sets: small disjuncts and class overlap [118].

Concept learners such as decision trees express the class label as a disjunctive (OR) list of conjunctions (AND) of feature values. For example, the label “Nice Day” could be computed as (Temperature = ”Warm” AND Rain = FALSE) OR (Temperature = ”Hot” AND Breeze = TRUE)\(^2\). For imbalanced data sets, disjuncts contain very few elements since there are very few minority cases. Small disjuncts make a classifier likely to overfit [55][117]. Hence oversampling or undersampling becomes necessary.

\(^2\)http://storm.cis.fordham.edu/ gweiss/small_disjuncts.html
Class overlap happens in most real-life data sets. In imbalanced data sets, the overlap zone is dominated by majority cases, impeding classifiers’ effectiveness in this crucial area [12]. Over-sampling techniques aim at solving this issue. One of these techniques is SMOTE (Synthetic Minority Oversampling Technique) [19]. SMOTE addresses the class overlapping problem in creating new minority cases between two minority nearest neighbors (see Figure 5.3.1). These synthetic instances are then added to the training set to reinforce a minority zone. The number of nearest neighbors to consider as well as the number of minority cases to create are tuning parameters.

While SMOTE may succeed for well-defined minority zones, applying SMOTE on relatively noisy data sets may lead to actually generating noise along the decision boundary [79] or even within the majority class [12].

Some undersampling techniques have been introduced in the late 1960’s [49][111] and can be used as boundary-cleaning methods too (in which case they also undersample the minority). The first use of Tomek’s technique appeared in 1997 in [71]. Since this method is based on nearest-neighbor, it is either too computationally-demanding or

Figure 5.1: Positives (red) are oversampled by SMOTE to generate synthetic positives (orange) and reinforce minority zones.
too sensitive to noise, according to [12]. Still in [12], more efficient methods combining SMOTE and undersampling are evaluated using a decision tree classifier. A more recent approach uses random forests to clean the majority class of the data set from overlapping zones, and then apply a random forest biased towards minority [42]. In some cases, SVMs have been shown to work well with drastic undersampling [96].

A random forest approach called Balanced Random Forest (BRF) uses perfectly balanced bootstraps (their size being twice the size of the minority) [22]. Each tree is given an undersampled version of the data set, but at the forest level, all majority cases have a chance to be picked by one or more trees. Hence there is no sharp undersampling, but rather an overall oversampling. This approach was extended in [53] where the authors replaced a fixed-size bootstrap by a bootstrap where the number of positives if fixed, but the number of negatives follows a negative binomial law. This method is called RBRF, for Roughly Balanced Random Forest.

5.3.2 Cost–sensitive Learning

Boosting: Some works have shown Adaboost’s weakness when data sets are imbalanced [64] or noisy [15]. Several boosting algorithms tailored for imbalanced data have been proposed, based on a normal oversampling [43], SMOTE [21] or Support-Vector Machines [79]. A recent work inspired by BRF builds T perfectly balanced samples of size twice the minority size, builds boosted classifiers on each of these balanced samples, and sums these T classifiers [78].

Cost–sensitive techniques applied to DTs and RFs: [22] proposed Weighted Random Forests (WRF) as an RF-based solution against imbalance. For each tree of the forest, weights are introduced in the criterion for finding splits at the node level, as well as when taking the “weighted” majority vote at the leaf level. Class weights are
parameters to be tuned. WRF results are reported to be similar to its re-sampling counterpart previously mentioned, BRF [22]. This empirical result had been predicted theoretically in [29], which showed that “splitting criteria (for DTs) in common use are relatively insensitive to costs and class distribution”. They also argued, based on [93], that class probabilities (“priors”) and misclassification costs are interchangeable in binary classification problems. Therefore WRF and BRF seem to be equivalent.

5.3.3 Decision Tree

Decision trees (DT) have been commonly used in the imbalance machine learning community. Yet, a 2004 workshop from this same research community reports a probably excessive reliance on C4.5, one of the most used decision tree algorithm [20]. Previous work also mentioned that using DT is inappropriate for fragmented or stratified data sets – data sets where using a divide-and-conquer approach is not effective, as illustrated in figure 5.2. Moreover, DTs are concept learners, hence they are subject to the “small disjunct” problem mentioned by Weiss [118] (see next section). Moreover, branches useful to predict the minority are likely to be pruned since replacing them by a majority leaf can lead to a lower error rate. A potential solution to this insidious and biased pruning has been recently proposed in [77].

Each classifier has its limitations, and we fully recognize those of decision trees. However, DTs are quite robust against noise and outliers, which occur in the PAN data set and in real-life data sets more generally. In our data set, our features showed little fragmentation (if at all), hence the effectiveness of DTs should not be reduced for that reason. We like using DTs also because they are quite visual, making them easier to understand and “debug” than logistic regression or neural networks.
5.3.4 Random Forest

Random Forest (RF) uses a bagging (for “bootstrap aggregating”) of collections of random Decision Trees. Each tree is built using a subset of the whole feature set (usually $|\text{subset}| = \sqrt{|\text{featureset}|}$) and using a bootstrap obtained from randomly picking data cases with replacement from the training set. The idea behind bagging is that each weak yet independent tree will make errors that other trees will not make. This also could partially solve the problem of stratified data. At test time, each tree classifies a test case. The predicted label is obtained from a majority vote from the entire forest. RF stays robust against noise and outliers and overfits less than a single tree [15].

5.4 Methods

We split the PAN training data set into 3 folds, and employed 3-fold cross-validation to observe the performance of our classifiers. Each fold contained roughly 5000 cases.
The test set was never merged with our train set, but rather we kept it for the very end to apply the best classifier we could obtain from the 3-fold cross-validation phase.

Our experiments were conducted using the Java framework Weka [47]. We extended several classes of the aforementioned classifiers from Weka so that we could measure their performance and compare them to each other. Because we wanted our solution to be practical, we also measured the time taken to build a solution as well as the time taken to run it on a validation set.

We wanted to be able to compare the performance of different classifiers. Hence we needed several runs of the same experiments on each of the folds to be able to significantly assess which was doing best. Building from scratch enough random forests to achieve significance would have required a number of random trees several order of magnitudes larger than the number of data cases. We decided to build a large pool of 10,000 random trees for each learner or configuration to be compared. Each run of a configuration or learner picks each of its 2000 trees randomly and without replacement from the pool of trees. 2000 is a small enough number compared to 10,000. The number of possible combinations of forests of size 2000 using 10,000 trees remains high \(10^{2171}\). It is unlikely that two forests of size 2000 contain many trees in common. During the forest construction process, *i.e.* for trees ranging from 1 to 2000, we logged the number of trees in the forest as well as the forest’s current performance.
5.5 Handling the Class Imbalance Problem for Random Forests

5.5.1 Seven Configurations

A total of seven configurations were run. We compared three types of Random Forests: Normal Random Forest (NRF) [15], Balanced Random Forest (BRF) [22], and Roughly Balanced Random Forest (RBRF) [53]. NRF was used as our “golden standard” and we did not alter its bagging procedure. For BRF and RBRF, we used three different bootstrap sizes:

- **SMALL**: bootstrap size was twice the minority size, *i.e.* around 1200 cases: 600 positives (all of those present in the training set) and (exactly for BRF, roughly for RBRF) the same number of majority cases, picked randomly with replacement from the training set. This configuration can be seen as a drastic undersampling of the majority.

- **SAME**: bootstrap size was around 9400 cases, *i.e.* the same size as NRF bootstrap: half of the majority cases (4700 of them) and (exactly or roughly) 4700 positives. Both positives and negatives were picked randomly with replacement. This approach can be seen as undersampling and oversampling combined.

- **DOUBLE**: bootstrap size was around 18,800 cases, *i.e.* nearly twice the size of NRF bootstrap: all of the 9400 negatives from the training set were picked once and only once, and 9400 positives were picked randomly and with replacement from the training set. Drastic oversampling only (all majority cases were in the bootstrap).
5.5.2 Performance measures

Each time a tree was added to a forest, several measures were reported:

- forest type with bootstrap size (NRF, BRF_SMALL, RBRF_SMALL, BRFSAME, RBRFSAME, BRF_DOUBLE, RBRF_DOUBLE)
- fold number, ranging from 1 to 3
- run number, ranging from 1 to 20
- number of trees in the forest, ranging from 1 to 2000
- number of True Positives (TP)
- number of False Negatives (FN)
- number of True Negatives (TN)
- number of False Positives (FP)
- AUC (Area Under the ROC Curve), calculated following the fast and efficient algorithm described in [32]

Logging TP, TN, FN and FP allowed us to plot the evolution of sensitivity, specificity and AUC; where Sensitivity (Acc+) is $\frac{TP}{P}$. It is also called True Positive Rate (TPR). Specificity (Acc-) is $\frac{TN}{N}$. It is also 1 - False Positive Rate.

5.5.3 Results

Graphs obtained from post-processing the log file can be seen in Figure 5.3. These graph lead us to the following conclusions.
First, and most strikingly, sensitivity and specificity remained constant independently of the forest size. NRF was outperformed by all others in terms of sensitivity (i.e. many false negatives) but was unmatched in terms of specificity (fewer false positives). Looking at the six other configurations, larger bags achieved higher specificity but lower sensitivity. Note that the ranges in sensitivity and specificity differ considerably: 0.6 to 0.9 for sensitivity versus 0.92 to 0.99 in specificity. This difference in ranges can be explained by the oversampling:

- Small bags contained all the minority examples, without resampling, and an undersampling of majority cases. Hence the classifier gave the same consideration to positives and negatives. This results in sensitivity and specificity being roughly equivalent, around 0.9 both.

- Medium-sized bags “duplicated” minority cases. Hence the classifier had around twice more actual information about positives than negatives. The classifier got biased towards majority cases (they are easier to classify). Specificity increased at the expense of sensitivity. This effect was even more striking for double-sized bags.

AUC increased with the number of trees slower for small–bags RF than for medium–, big–bags RFs or even normal RF. We think this is due to bags containing less actual information than what NRF bags contained. In BRF\_DOUBLE and RBRF\_DOUBLE bags, (nearly) all majority cases were represented. Hence they performed on the same scale as NRF. Small forests did not contain redundant information and gave the same importance to unique positives and negatives. We use the term unique here because the redundancy caused by oversampling in same- or double-size bags largely shadows the redundancy caused by bootstrap sampling with replacement. Bootstraps are built with replacement. A bootstrap of the same size as the train set contains, on average,
Table 5.1: Paired t–tests results (the table is symmetric).

<table>
<thead>
<tr>
<th></th>
<th>NRF</th>
<th>BRF_SMALL</th>
<th>BBRF_SMALL</th>
<th>BRFSAME</th>
<th>RBRF_SAME</th>
<th>BRF_DOUBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRF_SMALL</td>
<td>1,1,-1</td>
<td>0,0,0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BBRF_SMALL</td>
<td>1,1,-1</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
</tr>
<tr>
<td>BRF_SAME</td>
<td>1,1,-1</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
<td>0,0,0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BBRF_SAME</td>
<td>1,1,-1</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
</tr>
<tr>
<td>BRF_DOUBLE</td>
<td>1,1,0</td>
<td>-1,0,1</td>
<td>-1,0,1</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
</tr>
<tr>
<td>BBRF_DOUBLE</td>
<td>1,1,0</td>
<td>-1,1,0</td>
<td>-1,1,0</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
<td>-1,1,1</td>
</tr>
</tbody>
</table>

63% of the data cases in this train set. As trees are added to a forest, the number of negatives “seen” increases. Big bags simply brought negatives faster than small bags, hence for a large enough number of trees, big-bags RFs outperformed smaller-bags RFs. Ultimately, no re-sampling method discussed in this section did better than NRF for a large number of trees.

Each forest size from a particular forest type with a particular bootstrap size was run 60 times (20 runs and 3 folds). This large number of runs allowed us to reduce the standard deviation in measuring the classifiers’ performances and conduct a series of t–tests between classifiers. Results of the two–tailed paired t–test are presented in Table 5.1.

The goal of those t–tests is to statistically confirm or reject what is visible in Figure 5.3: the optimal random forest classifier varies with the number of trees. Therefore, each cell of table 5.1 contains a triple (a, b, c), where a stands for the result of a t-test for a random forest with 5 trees, b for 100 trees, c for 2000 trees. a, b and c take values in −1, 0, and 1, respectively for “AUC significantly lower”, “no significant difference in AUC”, “AUC significantly higher”. The level of significance we take is $p < 0.05$.

Example: cell (BRF_DOUBLE, BRF_SMALL) contains −1,0,1. This means that BRF_SMALL gave significantly higher AUC than BRF_DOUBLE for 5 trees, not significantly different for 100 trees, and significantly lower for 2000 trees.

Table 5.1 confirms that for forests with few trees (first and second values of the triple in each cell of the first column), NRF is outperformed. However, the takeaway is NRF outperforms or is comparable to other forests for large number of trees.
Figure 5.3: Mean AUC, sensitivity, and specificity for the seven random forest configurations. Both ends of the error bars are one standard deviation distance from the mean.
Figure 5.4: Average over the 3 folds of the building and testing times for 20 runs of our 7 random forest configurations. 10,000 trees were built. For each run, we only kept 2,000 trees in a random forest and classified a validation fold.

As Figure 5.4 shows, we observed an increase in training time linear with the bootstrap size. As expected, the training times for NRF, BRF\_SAME and RBRF\_SAME were comparable (same bootstrap size).

Figures 5.3 and 5.4 combined also show that for small forests, BRF\_SMALL or RBRF\_SMALL, as described in [22] and [53], outperform NRF in both training time and AUC. The space required in memory to store the classifier, however, is the same independently of the bag size: the trees in random forests were all limited to use sqrt(66) features, and random trees were never pruned.

5.6 SMOTE

SMOTE (Synthetic Minority Oversampling TEschnique) is an oversampling technique well–known within the data imbalance community. SMOTE is a pre–processing technique that takes two parameters: $k$ and $r$.

The parameter $k$ is the number of nearest neighbors from the minority to consider
Figure 5.5: SMOTE on stratified (left) or radial (right) imbalanced data is not always appropriate. Positives (red) oversampled by SMOTE can bring samples that densify the minority zone(s) (orange) or add noise (purple).

when oversampling, and $r$ is the rate of oversampling. For example, $SMOTE(5, 200)$ will triple the minority size in the training set (100% existing + 200% synthetic). The algorithm will randomly pick a positive data case and generate a synthetic positive data case between this point and one of its 5 closest positive neighbors.

SMOTE is particularly useful to densify minority zones; however, it is not as efficient for stratified or radial data sets, as shown in Figure 5.5.

For cases such as stratified or radial data sets, using a kernel before SMOTE quickly comes to mind. Projecting in kernel space before SMOTE is a viable alternative for small data sets, but medium to large data sets like ours would lead to very high dimensional spaces, which require a large amount of computation. This is not our goal.

Since the PAN data set was not noticeably stratified or radial, we decided SMOTE was applicable. The number of configurations to run for tuning grows quickly:

- we still wanted to measure the increase in performance metrics with the number of trees, up to 2000
- for statistical comparisons, each experiment was run 10 times, on each of the 3
folds. We generated 10,000 random trees as described in the previous experiment in order to gain time.

- k, the number of nearest neighbors, has to be tuned: we chose k in 1, 3, 5, 7, 9, 11, 13, 19, 25, 31, 37. Higher values of k become meaningless, as each training set only contained around 600 minority cases.

- r, the amount of over-sampling to perform also has to be tuned. We picked r in [0,1300] by steps of 100. A rate of 1300 corresponds to a nearly balanced training set containing 17,000 samples. A rate of 0 is equivalent to No–SMOTE, whichever k chosen. Hence, we have 11 times more “No–SMOTE” than actual SMOTE configuration runs. This should only affect the standard deviation of the metrics and the smoothness of their plot.

A single machine would have to build around 5 million random trees, bundle some of them in the 5,000 random forests, and test those random forests. On our commodity machine, building a single random forest of that size without SMOTE pre–processing and with generating exactly 2000 trees took around 4 minutes. Note that a higher oversampling rate means a larger training set, and a longer training time – up to 10 minutes for a rate of 1300! This whole tuning phase would have required our commodity machine around 20 days straight. We had to use a distributed technique like Map–Reduce to parallelize our tasks.

5.6.1 Map–Reduce on EC2

Map–reduce is a framework for large–scale data processing introduced by Google engineers in 2004 [27]. A map–reduce job first consists of a split into “configurations” of the large task to execute. Each configuration is assigned to a mapper. All mappers
have the same code, only the configuration they are given differs. During its task, a mapper keeps sending results to reducers. Reducers merge and sort results from the mappers. In the end, each reducer flushes into a file a sorted and/or merged list of the results it was given by the mappers.

Hadoop\(^3\) is an open-source implementation of the map-reduce framework by Apache that we have used to run our SMOTE configurations. We used 12 machines from the Amazon Elastic Compute Cloud.

The 5000 SMOTE configurations were built by a node called “master node”. A configuration was a tuple \((f, k, r)\) where \(f\) was the fold number, \(k\) the number of nearest neighbors to consider and \(r\) the amount of oversampling in SMOTE. The master node then sent configurations to each of the 20 available mappers. These mappers pre-processed their training data with \(\text{SMOTE}(k,r)\), built 10000 trees on this new training set, built 10 random forests of size 2000, and tested these 10 random forests on the fold number mentioned in the configuration. Mappers sent to reducers the configuration they were in charge of as well as the number of true positives, true negatives, false positives, false negatives, and the AUC they obtained during the testing phase. Seven reducers were in charge of merging and sorting the results.

The final output consisted of seven files (one per reducer), each of size 150MB, that is to say, a gigabyte. The post-processing of the map-reduce output data was conducted in six hours using the statistical environment R on our commodity machine.

5.6.2 Results

Figures 5.6 and 5.7 show that after 50 trees, most of the positives and negatives have been included in at least one of the trees of the forest. Therefore improvements in

\(^3\)http://hadoop.apache.org/
Figure 5.6: Sensitivity plots for oversampling rates of 100 and 1300.
Figure 5.7: Specificity plots for oversampling rates of 100 and 1300.
sensitivity or specificity are quite meager after 50 trees.

In terms of sensitivity, the no–SMOTE configuration (i.e. rate = 0) was outperformed by all actual SMOTE configurations (0.62 against up to 0.71 for rate = 100), but outperformed all others in terms of specificity. Presumably, a higher oversampling rate shifted a classifier’s focus towards minority and improved sensitivity at the expense of specificity. Also, applying SMOTE on our data set might have generated slightly more noise among negatives as it generated actual information in positives. This could explain why the AUC of all SMOTE configurations decreased with the oversampling rate, as seen in Figure 5.8.

Interestingly, large values of k brought higher sensitivity and lower specificity than small values of k. As the oversampling rate increased, the range of sensitivity covered by the different values of k stretched. For large values of k such as $k = 37$, sensitivity jumped from 0.71 for $r = 100$ to 0.78 for $r = 1300$. Plots of small values of k, however, moved closer towards the No–SMOTE boundary curve for high oversampling rates. Only plots for $k = 13$ remained constant around 0.69 with different rates. Moreover, for any given rate, no clear AUC distinction could be drawn between different values of k. More research is needed to explain this stretch centered around the $k = 13$ curve.

## 5.7 Experiment 3: cost–sensitive Learning

The performance of a classifier for a two–class problem can be described by the confusion matrix described in Table 5.2. The cost matrix will provide the costs associated with the four outcomes shown in the confusion matrix, which we refer to as $C_{TP}$, $C_{FP}$, $C_{FN}$, and $C_{TN}$. As is often the case in cost–sensitive learning, we
Figure 5.8: AUC against forest size plots for oversampling rates of 100 (top) and 1300 (bottom).
assign no costs to correct classifications, so $C_{TP}$ and $C_{TN}$ are set to 0. Since the positive (minority) class is often more interesting than the negative (majority) class, typically $C_{FN} > C_{FP}$ (note that a false negative means that a positive example was misclassified).

We can formulate the total cost as [118]:

$$TotalCost = (FNC_{FN}) + (FPC_{FP})$$

To implement cost-sensitive learning we use random forests. We set $C_{FP}$ to 1 and change the $C_{FN}$ from 1:14.5, where 1:14 is the imbalance rate in our data set. When we consider $C_{FN}$ as 1, the classifier behaves like a normal random forest. We did this experiment several times with forests with different numbers of trees \{1,2,4,8,16,\ldots,512,1024,2048\}.

To compare the AUC values for normal random forests ($C_{FN} = 1$) and AUC values of ones with higher ($C_{FNs}$), we use two-tailed paired-test (alpha\(\leq\)0.05) on test set. Table 5.3 shows two pairs of AUC values for $C_{FN} = 1$ and $C_{FN} = 10$.

According to these two pairs of AUC values, null hypothesis is rejected. It shows that AUC values for $C_{FN} = 10$ are significantly better compared to $C_{FN} = 1$. However, it is not true for most of random forests with $C_{FN}$ bigger than 1. Changes in AUC values as $C_{FN}$ increases are not significant.
Table 5.3: AUC values for two set of pairs: $C_{FP} = 1$ & $C_{FP} = 10$.

<table>
<thead>
<tr>
<th>Number of Trees</th>
<th>$C_{FP} = 1$</th>
<th>$C_{FP} = 10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8205</td>
<td>0.8347</td>
</tr>
<tr>
<td>2</td>
<td>0.8630</td>
<td>0.8907</td>
</tr>
<tr>
<td>4</td>
<td>0.9064</td>
<td>0.9208</td>
</tr>
<tr>
<td>8</td>
<td>0.9302</td>
<td>0.9341</td>
</tr>
<tr>
<td>16</td>
<td>0.9394</td>
<td>0.9442</td>
</tr>
<tr>
<td>32</td>
<td>0.9451</td>
<td>0.9473</td>
</tr>
<tr>
<td>64</td>
<td>0.9520</td>
<td>0.9500</td>
</tr>
<tr>
<td>128</td>
<td>0.9523</td>
<td>0.9542</td>
</tr>
<tr>
<td>512</td>
<td>0.9551</td>
<td>0.9550</td>
</tr>
<tr>
<td>1024</td>
<td>0.9561</td>
<td>0.9558</td>
</tr>
<tr>
<td>2048</td>
<td>0.9551</td>
<td>0.9561</td>
</tr>
</tbody>
</table>

5.8 Conclusion

Using a real-life application – detecting vandalized Wikipedia pages – we demonstrate that random forests are a remarkable classifier to use in and of itself. When the tuning of resampling techniques is possible, as it is often the case, we warmly recommend doing it, as AUC values can change drastically.

Similar to [29], we provide empirical evidences that resampling techniques can be equivalent to cost–sensitive techniques when using random forests as a classifier. Our approach also shows that successful results can be achieved on commodity machines in no time thanks to map–reduce: effectiveness does not exclude efficiency.
Chapter 6

Handling Missing Data Problem

6.1 Summary

One of the challenges facing social networks is vandalism, i.e., edits that damage content quality. The high visibility and easy access to social networks makes them popular targets for vandals. Detecting and removing vandalism is critical for these systems. Because vandalism can take many forms, there are many different features for detecting it. Recent studies show that user reputation features are important signals for vandalism detection. In social networks like Wikipedia, user reputation features can be calculated by mining user action histories. However, due to the large amount of missing data on user action histories, the vandalism detection system needs to cope with incompletely observed user reputation features. In this chapter we study the problem of missing data for user reputation features for the application of vandalism detection in Wikipedia. We try a variety of methods to handle missing data, based on several binary classifiers, and compare their appropriateness and classification performance. We show that, for some classifiers like logistic regression, data
imputation methods improve classification performance. However, for random forest, imputing missing values with a unique–outrange value yields the best classification results. We show that on the PAN 2010 corpus this approach results in $AUC = 0.9570$, which achieves a higher classification performance than all others previously reported.

### 6.2 Introduction

In many applications of machine learning, we need to build learning models and make predictions based on data sets with incompletely observed features. Sometimes, values are missing due to errors, omissions or other unknown reasons; especially when data are recorded and transferred from different sources. Features may be collected from sources incapable of providing the same set of input for all instances. Thus it results in data sets with missing information.

One example is very prevalent in Wikipedia: the deleting of poor quality articles along with their history revisions. This results in missing data for the users who contributed to those articles\(^1\).

In general, analysis of social network data is often hampered by non-responsive or missing data. It is often inferred, in models for social networks, that the presence or absence of data such as links in the network is completely observed, that the information is completely reliable and that there are no measurement errors. This is clearly not true in practice, as much network data is collected through partial crawling or online sample surveys [69, 48].

Recent studies show the negative effects of missing data concerning users and their action histories in social networks. This means that the results of social network\(^1\)http://en.wikipedia.org/wiki/Wikipedia:Articles_for_deletion
analyses can be severely biased if missing ties were ignored and only complete in-
stances were analyzed [48]. One such example is vandalism detection, where it is
helpful to augment the features with user–related features. User reputation features
are extracted from user action histories, while this data is not available for new users
or users with deleted action histories.

In our previous work [62], we showed the importance of user reputation features for the
purpose of vandalism detection. We showed that removing user reputation features
results in a significant drop in the classification performance. Since, for a significant
number of users, we do not have enough action histories to accurately calculate the
values for user reputation features, the problem of missing values needs to be handled
when applying classification methods. Although missing data has been studied in the
context of social networks before, little research has been conducted on its possible
effect on the application of spam filtering or vandalism detection.

In this chapter, we study the problem of missing data in the effort to detect van-
dalism. We discuss a variety of methods to handle missing data and compare their
appropriateness and classification performance based on PAN 2010 corpus [81]. We
use popular binary classifiers used widely for spam detection such as Naive Bayes,
logistic regression, SVMs, and random forests.

The contribution of this chapter is twofold. First, we show that the classification
performance of these four classifiers can be improved using data imputation methods.
Secondly, we show that random forests are robust classifiers and the imputation meth-
ods result in negligible fluctuation in their classification performance. Using random
forests, AUC is 0.9570 which is the best reported in the PAN 2010 corpus [90].

We show that, in general, the first three binary classifiers are sensitive to datasets
with missing values and in some cases data imputation can result in some significant
improvements in classification performance. However, random forests are robust classifiers with negligible fluctuation in their performance after applying data imputation methods.

This chapter is structured in the following way: we begin with a review of the relevant work on handling missing data in the literature. In subsequent sections we apply different missing data methods and study the behavior of different binary classifiers based on each method. In concluding, we argue the effectiveness of the methods and the performance of the classifiers.

### 6.3 Background

The issue of missing data has been studied extensively in machine learning literature. Little and Rubin [98] identify scenarios for missing values, pertaining to dependencies between the values of features and the absence of features. According to this study, there are three mechanisms of missing data: missing completely at random (MCAR), missing at random (MAR), and missing not at random (MNAR).

A feature is MCAR if the probability of its being absent is the same for all instances. If data is missing in such a way as to be completely at random, then discarding the instances with missing data does not bias the inferences. Most instances of missing data are not completely at random, as can be seen from the data themselves. For example, the different nonresponse rates in a certain Social Indicators Surveys for whites and blacks indicate that the “earnings” question is not missing completely at random. The general assumption is that it is MAR. MAR means that the probability of a feature missing depends only on other available features. Thus, if sex, race, education, and age are recorded for all the people in the survey, then “earnings” is
missing at random if the probability of nonresponse to this question depends only on these other features [39].

Missing data is MNAR if the feature with missing value depends on some features that have not been recorded and those feature can predict the missing values. For example, suppose that people with degrees in higher education are less likely to reveal their earnings, using an advanced degree as a feature can enable us to predict earnings. Then, earnings are not missing at random.

Another familiar example is taken from medical studies. In this case, if a particular treatment causes discomfort, a patient is more likely to drop out of the study. Instances of missing data in this case are not at random (unless “discomfort” is measured and observed for all patients). If the missing data is not at random, it must be explicitly modeled, or else some bias must be acknowledged in the inferences [39].

Most studies assume that missing values occur completely at random and thus various “imputation” methods have been designed to predict the missing values before building models. This scenario may not always prove true, however, during practical applications [41]. Nonetheless, it is a commonly assumed scenario that should be understood before moving to other analyses, especially since most imputation methods rely on MCAR for their validity [50]. Furthermore, Ding and Simonoff show that the performance of missing-value treatments used when training classification trees seems unrelated to the Little and Rubin taxonomy, as long as various instances of missing data do not depend on the class value (in which case unique-value imputation should be used, as discussed below, as long as the same relationship will hold in the prediction setting) [99].

In this work, we use binary classification based on 66 features where user reputation features suffer from missing values. We explore various reasons for the missingness
and try different data imputation methods, based on several binary classifiers, in order to compare their performance for the purpose of vandalism detection in Wikipedia.

### 6.4 The Problem of Missing Data

To effectively address the issue of missing data in our data sets, we need to know the reason for its missingness. In practice, one way to detect a correlation between the missingness of data and its class value is to try building a model, with a classification tree being a natural choice, of the class value on the missing features (which equals to “-1” if the corresponding class value is missing and “+1” otherwise). If such a model supports a relationship, then it is an indication that the missingness is related to the class value [125].

To this aim, we focus on the features that suffer from missing data: DSR, DDSR, and Rep. According to our data sets, 614 out of 14996 instances in train set and 796 out of 17443 in the test set are incomplete; so the rate of missing data is about 4% in each data set.

In this section, we analyze the correlation between instances with missing data and the corresponding class values. We consider the value of each feature as “-1” if the value is missing, and “+1” otherwise. Then, we train a binary classifier based on the three features and measure the classification performance based on ROC–AUC metric. If there is no correlation between the features and the label, we expect AUC value be around 0.50. Otherwise, AUC value deviates from 0.5.

For this study, we use Weka implementation for the four binary classifiers [46]. For SVM we use Weka LibSVM with RBF kernel. To tune the parameters of the classifiers we use a distributed approach based on MapReduce as explained in [35].
Table 6.1: Correlation analysis between reputation features and class values.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AUC (CV)</th>
<th>AUC (Test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.5000 ± 0.0000</td>
<td>0.5000</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.5084 ± 0.0026</td>
<td>0.4990</td>
</tr>
<tr>
<td>Random Forest (1K trees)</td>
<td>0.5084 ± 0.0026</td>
<td>0.4990</td>
</tr>
<tr>
<td>SVM</td>
<td>0.5022 ± 0.0103</td>
<td>0.4990</td>
</tr>
</tbody>
</table>

To learn a classifier and tune its free parameters, we use the PAN train set and keep the test set untouched for final evaluation.

Table 6.1 shows the classification performance for the binary classifiers. The first column shows the mean AUC along with its standard deviation on the train set based on 3-fold cross validation. The second column shows AUC values on the test set.

All AUC values for the four classifiers are about 0.50. According to these results, we assume that the missingness of the data is occurring at random and there is no correlation between the features with missing value and the corresponding class values. However, it is not possible to be sure whether data really are missing at random, or whether the missingness depends on unobserved features or the missing data itself. The fundamental difficulty is that these potential lurking features are unobserved by definition and so we can never rule them out. For this study, we try to include as many features as possible in a model so that the missing at random assumption is reasonable [39].

### 6.5 Handling Missing Data

Assuming that the missingness of data happens at random in both train and test sets, we try two main missing data methods: *discarding instances* and *data imputation*. 
In the rest of this chapter we use these methods to report the classification performance based on the binary classifiers.

To measure the effectiveness of each missing data method, we compare its classification performance (ROC–AUC) with that of the baseline. For baseline we impute the missing values with “-1”. The range of values for the user reputation features is between “0” and “1”, so we choose “-1” as an outrange value.

Having the AUC values for the baseline and each missing data method, we use the two–tailed paired t-test with p–value=0.05, to see if the difference is statically significant. For each experiment, we have 25 pairs: 24–fold cross validation based on train set and one on the test set. We have 4 different classifiers and we try 6 different missing data methods, and for each we run the experiment 25 times. In total, we run 600 experiments. Using Amazon’s MapReduce cluster, all experiments can be done in a few hours.

6.5.1 Discarding Missing Data

Many missing data approaches simplify the problem by throwing away the instances with missing data. Here we report the results based on Complete–case analysis: excluding all instances with missing data from the training set. We build a model based on the newly reduced train set and use the model for prediction.

Table 6.2 shows the AUC values for the baseline. The first column shows the mean AUC along with its standard deviation based on cross validation on the train set. The second column shows AUC value on the test set. The baseline results show that random forest outperforms the rest of the classifiers significantly.

---

The feature set used for these experiments is similar to the one introduced in 4. Except we
Table 6.2: AUC values for base line: imputing missing values with a unique outrange value

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AUC (CV)</th>
<th>AUC (Test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.9526 ± 0.0196</td>
<td>0.9132</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.9532 ± 0.0174</td>
<td>0.9153</td>
</tr>
<tr>
<td>SVM</td>
<td>0.9609 ± 0.0133</td>
<td>0.9333</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.9767 ± 0.0112</td>
<td>0.9566</td>
</tr>
<tr>
<td>(1K trees)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3: Discarding instances with missing values from train set.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AUC (CV)</th>
<th>AUC (Test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.9439 ± 0.1760</td>
<td>0.9080</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.9530 ± 0.0153</td>
<td>0.9252</td>
</tr>
<tr>
<td>SVM</td>
<td>0.9478 ± 0.0187</td>
<td>0.9146</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.9750 ± 0.0109</td>
<td>0.9547</td>
</tr>
<tr>
<td>(1K trees)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3 shows the AUC values after discarding the instances with missing values from the train set. Comparison of the AUC values in Table 6.3 and Table 6.2 shows that after discarding the instances with missing values, AUC values drop. According to the two-tailed paired t-test (p-value=0.05), the drop in AUC values is statically significant for Naive Bayes, SVM, and random forest.

It is interesting that discarding instances with missing data can result in a significant drop in classification performance even on this data set with 4% missing rate. Additionally, the handling of missing values in some popular machine learning softwares is mainly based on discarding instances with missing values. For example, Weka uses this approach to handle missing data for Naive Bayes and perceptron classifiers.

added one boolean feature which shows if a user is anonymous or registered.
6.6 Missing Data Imputation

Rather than discarding instances with missing data, another approach is to fill in or “impute” missing values. This approach is very popular when the missingness of data is occurring at random [50].

In general we can use a variety of imputation approaches that range from extremely simple to rather complex. These methods keep the full sample size, which can be advantageous for biased outcomes, as a result of using certain discarding approaches [39]. In this section, we study the effectiveness of some popular data imputation methods.

6.6.1 Mean Imputation

One common method of data imputation is to replace each missing value with the mean of the observed values for a feature. We replace the missing values of DSR, DDSR, and Rep by their means based on the train set. Figure 6.1 shows the distribution of user reputation values after removing the missing values. Mean of values for DSR is $0.5724 \pm 0.2882$. It is $0.6957 \pm 0.2778$ for DDSR, and $0.7373 \pm 0.2630$ for Rep.

Table 6.4 shows the AUC values after replacing the missing values with the means. Using a paired t-test, the mean imputation results in significant improvement in classification performance only logistic regression. For random forest and SVM the baseline AUC values are still significantly better. It is interesting to note that Weka uses mean/mode approach (mode for nominals and mean for numerics) for logistic regression, and SVM classifiers like LibSVM and SMO.
Figure 6.1: Distribution of user reputation features on train and test sets

For tree based methods, Weka split instances up when there is a missing value encountered in a feature that is used for a test. They use weighted average based on the percentage of train instances in the two final leaves. In other words, when a feature value is missing, it tries both branches from the split. Looking at the training data it calculates what percentages of instances ended up in class 1 and what percentage in class 0. Then, the weighted average is the probability used for prediction.

Because of the open editing model of Wikipedia, users can contribute anonymously or as registered users. Several studies have shown the difference between the behavior of these two types of users [59]. For example, anonymous users are more likely to
Table 6.4: Imputing by mean results.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AUC (CV)</th>
<th>AUC (Test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.95078 ± 0.0175</td>
<td>0.9140</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.9687 ± 0.0165</td>
<td>0.9409</td>
</tr>
<tr>
<td>SVM</td>
<td>0.9510 ± 0.0232</td>
<td>0.9155</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.9766±0.0109</td>
<td>0.9569</td>
</tr>
<tr>
<td>(1K trees)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.5: Imputing by mean results. Mean is calculated separately for anonymous and registered users.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Anonymous Users</th>
<th>Registered Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSR</td>
<td>0.4458 ± 0.3875</td>
<td>0.6250 ± 0.2096</td>
</tr>
<tr>
<td>DDSR</td>
<td>0.5480 ± 0.3806</td>
<td>0.7623 ± 0.1800</td>
</tr>
<tr>
<td>Rep</td>
<td>0.5914 ± 0.3691</td>
<td>0.8030 ± 0.1583</td>
</tr>
</tbody>
</table>

do vandalism [58, 114]. We analyze the values of user reputation features for both anonymous and registered users. Table 6.5 shows the mean and standard deviation of the values in the train set after removing missing values. We do mean imputation separately for anonymous and registered users. Table 6.6 shows the AUC values after separating the mean imputation for each set of users. Compared to the imputation that results when we apply the same mean for both types of users, this approach results in better AUC values. However, the paired t-test shows that the improvements are not statically significant.

Table 6.6: Results for imputing by separate means for anonymous and registered users.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AUC (CV)</th>
<th>AUC (Test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.95124 ± 0.0175</td>
<td>0.9145</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.9681 ± 0.1560</td>
<td>0.9416</td>
</tr>
<tr>
<td>SVM</td>
<td>0.9509 ± 0.0235</td>
<td>0.9143</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.9772 ± 0.0110</td>
<td>0.9562</td>
</tr>
<tr>
<td>(1K trees)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 6.7: Imputing by zero results.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AUC (CV)</th>
<th>AUC (Test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.9483 ± 0.0189</td>
<td>0.9123</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.9641 ± 0.1587</td>
<td>0.9336</td>
</tr>
<tr>
<td>SVM</td>
<td>0.9508 ± 0.0202</td>
<td>0.9198</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.9765 ± 0.0113</td>
<td>0.9570</td>
</tr>
<tr>
<td></td>
<td>(1K trees)</td>
<td></td>
</tr>
</tbody>
</table>

6.6.2 Zero Imputation

One common way to deal with missing data in questionnaires is imputing with zero. This approach is also widely used in online user based applications when there is no information about a specific user. In the case of Wikipedia, zero imputation for user reputation features means that, by default, we put no trust in users with unknown action histories. This assumption might be reasonable, especially when the reason for missing data is the deletion of poor quality content. In this case we are assigning a low reputation to users who have contributed low quality content.

Table 6.7 shows the classification performance after replacing missing values with zero. The results show that for three of the classifiers AUC values are higher compared to the baseline results. However, the paired t-test results show that the improvement is significant only for logistic regression but; it is statically significant for random forest and Naive Bayes. For SVM the base results are still significantly better.

6.6.3 Learning Missing Values

In this section we do data imputation by learning the missing values of the user reputation features based on 63 other features. To achieve this aim, we use linear regression which has been used widely in different applications[122, 124]. Using the Weka regression library, we learn DSR, DDSR. Table 6.8 shows the classification per-
Table 6.8: Imputing by learning results.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AUC (CV)</th>
<th>AUC (Test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.9511 ± 0.0175</td>
<td>0.9144</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.9692 ± 0.0162</td>
<td>0.9413</td>
</tr>
<tr>
<td>SVM</td>
<td>0.9548 ± 0.1960</td>
<td>0.9184</td>
</tr>
<tr>
<td>Random Forest (1K trees)</td>
<td>0.9767 ± 0.0116</td>
<td>0.9567</td>
</tr>
</tbody>
</table>

formance after imputing by the learned values. The results show that only in the case of logistic regression is the improvement statically significant, while for SVM baseline it is still significantly better. For random forest and Naive Bayes the improvement is not statically significant.

6.7 Conclusion

In this chapter we studied the issue of handling missing data for the purpose of vandalism detection. According to our data set about 4% of user reputation features suffer from missing values. We showed that there is no correlation between the features with missing values and the class label. Assuming that the missingness of data is occurring at random, we tried two main methods: discarding the instances and data imputation. We used four different binary classifiers: Naive Bayes, logistic regression, SVM, and random forest. We showed that the method of discarding the instances of missing data results in a drop in the classification performance. Using the two-tailed paired t–test with p–value=0.05, we showed that this drop is statically significant for random forest, SVM, and Naive Bayes.

We also tried several popular data imputation methods, and random forest with zero imputation resulted in the best classification performance on the test set, $AUC = 0.9570$ performing significantly higher than that of the winner of the PAN competition.
[81]. However, the improvement in the classification performance was not statically significant compared to that of the baseline. In the baseline, we impute the missing values with a unique and outrange value. In general, random forest showed to be a robust classifier and the baseline results could not be improved upon by any of the other data imputation methods. However, logistic regression showed to be very sensitive to missing data and classification performance fluctuated significantly.
Chapter 7

Categorizing Vandalism

7.1 Summary

In this chapter, we report on the construction of the Wikipedia vandalism categories based on PAN 2010 corpus, using Amazon’s Mechanical Turk. Based on the previous literature on the different types of vandalism in Wikipedia, we categorize vandalistic edits into eight categories. We explain how we collected the data and discuss the lessons learned. Furthermore, we use binary classifiers to classify vandalism in each of the categories. We measure the classification performance in each category. We show that the classification performance in some categories like edit error or comment vandalism is lower than the rest of the categories. Then, we discuss what groups of features are important for different categories. We show that user features are the most important features for six of the groups. For the two remained ones, textual features are better signals for vandalism detection.
7.2 Introduction

Vandalism detection has been a concern for Wikipedia since its inception. Vandalism in Wikipedia is defined as any addition, removal, or change of content in a deliberate attempt to compromise the integrity of Wikipedia. According to this broad definition, vandalism can include spamming, lobbying and destroying the edits of others. Wikipedia relies mostly on its human editors and administrators to fight vandalism. But the enormous magnitude of Wikipedia now makes locating all instances of vandalism a very time consuming endeavor. Tools such as Vandal Fighter, Huggle, and Twinkle are used to monitor recent changes in articles and to revert any of those changes that might be deemed vandalism [38].

There are different types of vandalism in Wikipedia. According to Wikipedia, all vandalism is serious because it affects the credibility of Wikipedia content. In this study, we investigate the types of vandalism in Wikipedia based on previous literature [92, 114] and explain how we can detect them automatically.

In the previous chapters we provided a machine learning solution to the problem of vandalism detection in general. We showed that the classifiers benefit from high accuracy. However, it is not clear how well does our system work on each of the vandalism types. Thus, in this chapter, we focus on common types of vandalism in Wikipedia, namely: misinformation, mass delete, partial delete, offensive, spam, nonsense, edit error and comment text vandalism. The focus of our concern is upon the automatic detection of different types of vandalism in Wikipedia, i.e., a systematic way of detecting the different types of vandalism that threaten to damage an article. We contribute to this research field by developing a collection of humanly–annotated vandalism edits, which is a prerequisite for any meaningful evaluation of the algorithms pertaining to vandalism detection.
7.2.1 Background

We describe two broad approaches to the thwarting of vandalism: the user approach, which relies mostly on tools to assist user detection; and the more automated approaches that generally rely on bot or other algorithms.

Viegas et. al. [114] conducted some early work on the types of vandalism found in Wikipeida. They used a visualization technique called “history flow” to see the various ways pages were edited and changed over time. In considering these changes they identified five types of vandalism: Mass Deletion, Offensive Copy, Phony Copy, Phony Redirection, and Idiosyncratic Copy. They also analyzed the time for repair of vandalism, which they termed “survival time”. They found that the median survival time for some vandalism is quite short, on the order of minutes. However, they noted that the mean time can sometimes be quite long—on the order of days or weeks—depending on the type of vandalism. This means that there is some vandalism that goes undetected for long periods of time.

In the followup work, Priedhorsky et. al. [92], considered the impact of a piece of vandalism. That is, if some vandalism lasts on a site for days, how likely is it that a user might stumble across that and be misinformed or otherwise get a wrong impression about the quality of the entire content based on a vandalized page. They developed a model based on page viewing behaviors and vandalism persistence. Their model correlates closely with the empirical results from Viegas et. al. [114] and further illustrates that about 11% of vandalism would last beyond 100 page views, with a small fraction of a percent lasting beyond 1000 page views. Further, Priedhorsky et. al. [92] also considered the types of vandalism and how likely users were to agree on the types of vandalism. They began with the vandalism types from [114] with some small refinements and identified two additional types: Misinformation, and Partial
Delete. They identified the most frequent categories of vandalism as Nonsense (Phony Copy) 53%, Offensive 28%, Misinformation 20% and Partial Delete 14%. However, they also noted that the rate of agreement among their users for some categories is somewhat low with Misinformation having the lowest level of agreement for these top four categories. This means that some of the most subtle vandalism, in the form of Misinformation, is even hard for users to agree upon.

These early attempts to understand types of vandalism in Wikipedia were done on small corpora. They thus had failed to account for the true distribution of vandalism types among all edits. Furthermore, the hand labeling of the examples had not been double-checked by different annotators. These shortcomings were resolved by the creation of a large-scale corpus for the PAN 2010 competition consisting of a week’s worth of Wikipedia edits (PAN–WVC–10) [90]. Using Amazon Mechanical Turk, Potthast constructed a PAN Wikipedia vandalism corpus. The corpus compiles 32,452 edits on 28,468 Wikipedia articles, among which 2,391 vandalism edits have been identified. 753 human workers cast a total of 193,022 votes on the edits, so that each edit was reviewed by at least 3 workers, whereas the achieved level of agreement was analyzed in order to label an edit as “regular” or “vandalism”. The corpus is available free of charge [81].

We did a study on the types of vandalism in Wikipedia based on the PAN data set. In this work, we build upon Potthast’s work [90] and categorize vandalistic edits in the PAN data set into eight categories. To this aim, we used Amazon’s Mechanical Turk. Amazon Mechanical Turk$^1$ is a crowdsourcing Internet marketplace that expedites projects requiring human intelligence by providing an interface between workers and project owners (requesters).

$^1$https://www.mturk.com/
7.3 Methods

We extracted 2,391 vandalism edits from the PAN Wikipedia Vandalism Corpus 2010 (PAN-WVC-10)[90]. Building upon previous research on vandalism types in Wikipedia [92] and considering the definition of vandalism in wikipedia website 2, we categorized vandalism into eight different categories:

- **Misinformation**: Information which is false or invalid, such as changed dates, inappropriate insertion of “not”, incorrect statements about public figures (Similar to the definition in [92]). We also consider modifying or redirecting a link to point to an inappropriate location, or hiding a potentially offensive link behind legitimate or otherwise harmless text. Note that this kind of information in many cases seems valid at first glance.

- **Mass delete**: Removal of all, or almost all, of an article’s content (Same as definition in [92]).

- **Partial delete**: Removal of a significant fraction of an article’s content, from several sentences to many paragraphs (Same as definition in [92]).

- **Offensive**: Text or imagery offensive to many users. For example, obscenities, hate speech, obvious attacks on public figures or other entities such as organizations, etc. This is a broad category, ranging from simple obscenities to outright pornography (Same as definition in [92]).

- **Spam**:Adding obvious advertisements, or links to commercial or other external sites. Note that links are usually located between “[” and “]” (Same as definition in [92]).

• **Nonsense:** Text that seems meaningless or unnecessary or irrelevant to the topic, [and technical markup leaking into formatted pages] (Same as definition in [92]).

• **Edit Error:** Text that appears to be mangled because of incorrect WikiText, HTML or other wrong markup; text that looks like it was saved in the middle of an editing session; text that appears legitimate but may be in the wrong section/location of an article. In general, any editing damage in the text fits into this category (No analogue in [92]).

• **Comment text vandalism:** Any kind of vandalism that has appeared in the comment section of an edit. Note that the comment on an edit is not a part of the text itself; it is located on top of it (No analogue in [92]).

We used Amazon’s Mechanical Turk to categorize the vandalistic edits into the aforementioned categories. We divided the entire work into 5 rounds. In each round we released 500 edits and requested six workers to specify the vandalism categories. Thus, we had a total of 3,000 tasks per round except for round 4 and 5 that we released more than 3000 tasks. We utilized a total of 196 worker, yielding 18,178 votes. We used plurality as our reviewing strategy; that is: if more than three of the workers agree upon a category, that category was selected. For each vandalistic edit, more than one category could be valid. For example, Figure 7.1 shows a vandalistic edit, which fits in both misinformation and comment text vandalism categories.

Since it was our first experience using Amazon’s Mechanical Turk, we did not use the service efficiently and effectively in the first rounds. A good task should retain workers by providing clear and understandable instructions and a convenient page to work on. In the first round, we put all of the instructions in the task page that ended

In the time of Girls Aloud year break the girls are working on their own individual projects. [[Cheryl Cole]] is continuing her role as a judge on [[The X Factor (UK)|The X Factor]] as well as working on her solo career. Her debut single [[Fight For This Love]] went straight to number one in the UK and Ireland selling 600,000 copies. As well as this, her debut album [[3 Words]] went number one in UK as well as going platinum. [[Nadine Coyle]] is working with major record company to create her solo album which will be released in 2010. [[Sarah Harding]] will be recording tracks for the soundtrack of [[St. Trinian's II: The Legend of Fritton's Gold]]. **It is widely acknowledged that Nadine and Nicola are the only good singers in the group.** Nadine is working on her solo career in Los Angeles, she is biding her time to ensure that her music is good quality rather than the nonsense released by fellow Girls Aloud member Cheryl Cole. Nicola has a very good voice and has stated she would like a solo career once Girls Aloud split up. She could be the next Christina Aguilera if she bides her time.

Figure 7.1: A vandalism example that fits in more than one category
Table 7.1: Properties of the 5 rounds

<table>
<thead>
<tr>
<th>Round</th>
<th>Creation</th>
<th>Completion</th>
<th>Duration</th>
<th>Reward</th>
<th>Qualification</th>
<th>Visibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>04/29/2011</td>
<td>05/07/2011</td>
<td>20 min.</td>
<td>$0.020</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>05/06/2011</td>
<td>05/10/2011</td>
<td>3 min.</td>
<td>$0.040</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>05/09/2011</td>
<td>05/17/2011</td>
<td>2 min.</td>
<td>$0.040</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>05/15/2011</td>
<td>05/17/2011</td>
<td>2 min.</td>
<td>$0.050</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>05/17/2011</td>
<td>05/18/2011</td>
<td>2 min.</td>
<td>$0.050</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 7.2: Progress the 5 rounds

<table>
<thead>
<tr>
<th>Round</th>
<th>Tasks</th>
<th>Completed</th>
<th>Daily Approved Tasks</th>
<th>Avg time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>D1</td>
<td>D2</td>
</tr>
<tr>
<td>1</td>
<td>2400</td>
<td>2105</td>
<td>137</td>
<td>514</td>
</tr>
<tr>
<td>2</td>
<td>3000</td>
<td>3000</td>
<td>8</td>
<td>230</td>
</tr>
<tr>
<td>3</td>
<td>3000</td>
<td>2298</td>
<td>344</td>
<td>663</td>
</tr>
<tr>
<td>4</td>
<td>4302</td>
<td>4295</td>
<td>773</td>
<td>3074</td>
</tr>
<tr>
<td>5</td>
<td>3564</td>
<td>3547</td>
<td>1259</td>
<td>2277</td>
</tr>
</tbody>
</table>

one more round. Using this approach, we still had 160 edits that were inconclusive.

Table 7.1 and 7.2 shows the specification of each round.

At the end of the process we discovered that comment text vandalism is not easy to detect. Only a few workers could detect it correctly; especially when the edit belonged to another category. We therefore came to the conclusion that plurality alone is not sufficient as a means of detecting Comment text vandalism. With that in mind, we extracted 1,537 vandalism edits, out of 2,391 which had comment text, and assigned them to two experts separately.

A total of 16266 tasks were released out of which 15245 tasks were completed and 32 tasks were rejected due to the low quality of some workers. Therefore, we ended up with 15213 approved tasks.

We evaluated the approved tasks. Altogether, 14,107 tasks were finalized which constitute 2,225 unique vandalism edits that reached consensus on one or two types of
Table 7.3: Vandalism Type Distribution Among Consensus Ones

<table>
<thead>
<tr>
<th>Type</th>
<th>Number of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonsense</td>
<td>1170</td>
</tr>
<tr>
<td>Offensive</td>
<td>508</td>
</tr>
<tr>
<td>Misinformation</td>
<td>437</td>
</tr>
<tr>
<td>Partial delete</td>
<td>137</td>
</tr>
<tr>
<td>Spam</td>
<td>57</td>
</tr>
<tr>
<td>Mass delete</td>
<td>54</td>
</tr>
<tr>
<td>Edit Error</td>
<td>45</td>
</tr>
<tr>
<td>CommentTextVandalism</td>
<td>20</td>
</tr>
</tbody>
</table>

vandalism. We had 196 human annotators with more than 18,178 votes that worked on 14,107 tasks with total time of 37,4017 seconds.

Table 7.4 shows the number of submitted answers (A) and answers with consensus (B) for the finalized tasks based on vandalism types.

The remaining 1,121 tasks have been set aside because their corresponding 160 vandalism edits could not be determined based on the submitted answers. Hence, our results focused on tasks which have been finalized. Table 7.3 shows the distribution of vandalism categories Among consensus ones. These results indicate that the Nonsense constitutes the majority of vandalism, which acknowledge the results of previous studies [86, 24].

\(^\text{4}\)The mentioned studies may call vandalism types differently.
### Table 7.4: Vandalism Type Distribution Among Workers’ Answers

<table>
<thead>
<tr>
<th>Type</th>
<th>Number of answers (A)</th>
<th>Answers with consensus (B)</th>
<th>Accuracy = B/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>18178</td>
<td>10887</td>
<td>0.59</td>
</tr>
<tr>
<td>Nonsense</td>
<td>7223</td>
<td>5742</td>
<td>0.79</td>
</tr>
<tr>
<td>Offensive</td>
<td>3378</td>
<td>2351</td>
<td>0.69</td>
</tr>
<tr>
<td>Misinformation</td>
<td>3774</td>
<td>1965</td>
<td>0.52</td>
</tr>
<tr>
<td>Spam</td>
<td>950</td>
<td>287</td>
<td>0.30</td>
</tr>
<tr>
<td>Edit Error</td>
<td>557</td>
<td>164</td>
<td>0.29</td>
</tr>
<tr>
<td>Partial delete</td>
<td>1163</td>
<td>322</td>
<td>0.27</td>
</tr>
<tr>
<td>Mass delete</td>
<td>574</td>
<td>144</td>
<td>0.25</td>
</tr>
</tbody>
</table>

### 7.4 Lessons Learned

#### 7.4.1 Be Specific

To obtain better results, the requester should design questions as specific as possible in order to minimize worker error. For example, in the first round we expected workers to distinguish vandalism from legitimate edits and then determine what type of vandalism it was. But the high number of FPR (0.28125) and FNR (0.06408) suggested that it was the mixture of the vandalism and not vandalism edits that was degrading the results. Therefore, in the remaining rounds, only vandalism edits were displayed and the workers would have to detect the type of vandalism only. In this paper, we have ignored 501 answers which were assigned to 100 unique edits in the first round that were determined not to be vandalism.
7.4.2 Be Visible

Based on our observation, visibility is an important parameter in speeding up the completion of tasks in Amazon Mechanical Turk. By visibility we mean the task of being available on the first page of the task list sorted according to criteria, such as HIT Creation Date, Time Allotted, Reward, etc. On the other hand, the first round helped us estimate the amount of time that a worker would need to accomplish a task. We then reduced the allotted time for the rest of the rounds in order to increase our visibility in the Time Allotted (least first) group. However, round 3 could not come up in the first page of the Time Allotted (least first) group due to the presence of too many tasks with less time allotted to them. As the reader can see in Table 7.1, rounds 2, 4 and 5, which were visible on the first page, were accomplished much faster than the other two rounds. The range of the time that a worker spent on a task varies between 9 to 324 seconds. However, workers who mostly succeeded in detecting the correct type spent less than 60 seconds. Therefore, allotting a larger time does not necessary lead to better results.

7.4.3 Be Rewarding

We discovered another important parameter for attracting workers to this task; namely: the monetary reward. When we increased the reward for rounds 4 and 5, they were finished very quickly.

7.4.4 Be Wary of Qualification Tests

Although using qualification tests has some advantages, it may discourage some workers. In rounds 2 and 3, only those workers who had passed the qualification test were
able to work on our tasks. But since the progress of round 3 was very slow, we decided to remove the qualification test for rounds 4 and 5.

### 7.5 Preliminary Results

In [62] we modeled vandalism detection as a binary classification problem, where the goal was to distinguish between vandalistic and legitimate edits. We used 66 features in four different categories: user features, textual features, meta data features, and language model features.

Using random forests, we showed that the classification performance in terms of ROC–AUC is 0.9570. However, this number shows the classification performance in general, and without regard to the vandalism type. We know that some types of vandalism are easier to detect. For example, offensive edits containing common vulgar words are much easier to detect than misinformation. Even for mere humans, such as ourselves, it is no small task to detect misinformation at first glance.

The ultimate question before us is: how well does our system work on each of these classes? To answer this question, we develop an independent binary classifier for each vandalism category and measure the classification performance. We tried several binary classifiers and discovered that, in all of our experiments, a random forest outperformed them all. Table 7.5 shows AUC values on a test set using random forests with 1,000 trees.

These initial results show that classification performance varies significantly among different vandalism categories. It is very high for some categories, like offensive and mass delete; but it is lower for some other categories, such as edit error and comment text vandalism.
Table 7.5: AUC values for the eight vandalism categories

<table>
<thead>
<tr>
<th>Vandalism Type</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonsense</td>
<td>0.9682</td>
</tr>
<tr>
<td>Offensive</td>
<td>0.9809</td>
</tr>
<tr>
<td>Misinformation</td>
<td>0.9406</td>
</tr>
<tr>
<td>Edit Error</td>
<td>0.8761</td>
</tr>
<tr>
<td>Mass Delete</td>
<td>0.9770</td>
</tr>
<tr>
<td>Partial Delete</td>
<td>0.9066</td>
</tr>
<tr>
<td>Comment</td>
<td>0.8850</td>
</tr>
<tr>
<td>Spam</td>
<td>0.9231</td>
</tr>
</tbody>
</table>

The reasons for why we have such variation needs more study; but we suspect that the lack of good features for edit error and comment text vandalism has resulted in lower classification performance. For example, the language of the actual text of a Wikipedia article is rather formal while its user comments are very informal. So this issue needs to be considered during the process of feature extraction. For the edit error category, mining the Wiki markup language and learning vandalism patterns seems necessary.

An interesting observation to be made is that of the AUC value for the misinformation category. We expected to see a lower classification performance for this category for the simple reason that it is humanly difficult to detect misinformation without considerable investigation.

To see the importance of different groups of features (see chapter 4 for definition of the groups), we report the AUC values based on two sets of experiments. First, we ignore a group of features and look at the amount of drop in the AUC values. Table 7.6 shows the AUC values after removing a specific group of features. The drop after removing user features is the highest for six types of vandalism such as misinformation, nonsense, and spam. For example, after removing these user features from the feature set, the classification performance drops from 0.9406 to 0.8329. However, for mass delete and comment text vandalism the drop in AUC reaches its maximum when we
Table 7.6: Classification performance after removing a group of features.

<table>
<thead>
<tr>
<th>Vandalism Type</th>
<th>All Features</th>
<th>User</th>
<th>Textual</th>
<th>Meta Data</th>
<th>LM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonsense</td>
<td>0.9682</td>
<td>0.9204</td>
<td>0.9476</td>
<td>0.9653</td>
<td>0.9669</td>
</tr>
<tr>
<td>Offensive</td>
<td>0.9809</td>
<td>0.9448</td>
<td>0.9589</td>
<td>0.9777</td>
<td>0.9818</td>
</tr>
<tr>
<td>Misinformation</td>
<td>0.9406</td>
<td>0.8329</td>
<td>0.9244</td>
<td>0.9307</td>
<td>0.9366</td>
</tr>
<tr>
<td>Edit Error</td>
<td>0.8761</td>
<td>0.7037</td>
<td>0.8855</td>
<td>0.8629</td>
<td>0.8838</td>
</tr>
<tr>
<td>Mass Delete</td>
<td>0.9770</td>
<td>0.9880</td>
<td>0.9244</td>
<td>0.9791</td>
<td>0.9720</td>
</tr>
<tr>
<td>Partial Delete</td>
<td>0.9066</td>
<td>0.8188</td>
<td>0.9375</td>
<td>0.8926</td>
<td>0.9065</td>
</tr>
<tr>
<td>Comment Text</td>
<td>0.8850</td>
<td>0.8464</td>
<td>0.8403</td>
<td>0.8491</td>
<td>0.8890</td>
</tr>
<tr>
<td>Spam</td>
<td>0.9231</td>
<td>0.8451</td>
<td>0.9248</td>
<td>0.9086</td>
<td>0.9108</td>
</tr>
</tbody>
</table>

Table 7.7: Classification performance after keeping one group of features.

<table>
<thead>
<tr>
<th>Vandalism Type</th>
<th>All Features</th>
<th>User</th>
<th>Textual</th>
<th>Meta Data</th>
<th>LM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonsense</td>
<td>0.9682</td>
<td>0.9356</td>
<td>0.8741</td>
<td>0.7003</td>
<td>0.6287</td>
</tr>
<tr>
<td>Offensive</td>
<td>0.9809</td>
<td>0.9391</td>
<td>0.9066</td>
<td>0.7131</td>
<td>0.6369</td>
</tr>
<tr>
<td>Misinformation</td>
<td>0.9406</td>
<td>0.9053</td>
<td>0.7377</td>
<td>0.6885</td>
<td>0.5606</td>
</tr>
<tr>
<td>Edit Error</td>
<td>0.8761</td>
<td>0.8669</td>
<td>0.6394</td>
<td>0.6482</td>
<td>0.4710</td>
</tr>
<tr>
<td>Mass Delete</td>
<td>0.9770</td>
<td>0.9129</td>
<td>0.9558</td>
<td>0.7111</td>
<td>0.9294</td>
</tr>
<tr>
<td>Partial Delete</td>
<td>0.9066</td>
<td>0.9070</td>
<td>0.7660</td>
<td>0.7413</td>
<td>0.7992</td>
</tr>
<tr>
<td>Comment Text</td>
<td>0.8850</td>
<td>0.7778</td>
<td>0.9070</td>
<td>0.5771</td>
<td>0.6863</td>
</tr>
<tr>
<td>Spam</td>
<td>0.9231</td>
<td>0.8983</td>
<td>0.7714</td>
<td>0.6293</td>
<td>0.6820</td>
</tr>
</tbody>
</table>

ignore textual features.

In the second set of experiments, we keep only one group of features and build the classifiers based on that group. Table 7.7 shows the AUC values. These results also show that user reputation features result in the highest performance for the six types of vandalism such as misinformation, nonsense, and spam; but for mass delete and comment text vandalism textual features result in the highest performance.

### 7.6 Conclusion

In this chapter we explained the process of constructing a multi-label Wikipedia vandalism corpus using Amazon’s Mechanical Turk. We categorized vandalism into eight common categories. For some edits, most of the workers in Mechanical Turk
consistently identified the vandalism according to its category. However, for a few categories, like edit error or comment vandalism, we did not observe such strong consensus. We therefore went on to use human experts in an effort to reach that consensus.

The ultimate goal of this study is to provide a vandalism detection system that is very accurate in each of the categories; especially the more common ones. In our initial experiments, we used independent binary classifiers to classify vandalism into separate groups. The results show that classification performance varies significantly among the eight categories. In addition, we investigated the importance of different groups of features for vandalism detection in each of the groups. We showed that for the six out of the eight groups user features are the most important features. For the two remained groups, mass delete and comment text vandalism, textual features contributed the highest to the classification performance.
Chapter 8

Conclusion

8.1 Conclusion and Future Work

In this thesis I studied the problem of measuring user reputation and content quality in Wikipedia. I modeled user reputation in Wikipedia and showed the effectiveness of three models for predicting user behavior. I used these reputation models to estimate content quality in Wikipedia. I showed that content quality is highly correlated to the reputation of its authors. I then extended the approach to enable me to locate low-quality content in Wikipedia in the form of vandalism. I described a machine learning approach to detect vandalism in Wikipedia.

I modeled vandalism as a binary classification problem. Based on the validated corpus of Wikipedia edits from the PAN 2010 competition, I trained and tested several binary classifiers utilizing learning methods that have been widely used for spam detection, such as: Naive Bayes, Logistic Regression, and SVMs. I also trained models using random forests which have not, as yet, been widely used for spam detection. Interestingly the results show that the random forests significantly outperform the
other three types of classifiers on this problem. An additional benefit of random forests is that they are robust to missing and unbalanced data which is a common characteristic of vandalism/spam data sets. To the best of my knowledge, my work provides the best classification results (AUC = 0.9570) on vandalism detection based on PAN 2010 test corpus.

A common practice for spam detection has been to train the models using all available features. Because my goal is to develop models that are small and fast enough to be used in a real-time vandalism detection tool, I used feature selection to eliminate redundant features so that the computational complexity of the final model is as small as possible while retaining high accuracy. Some of the features that proved to be most informative require the mining of user edit patterns.

Computing the value of these features can be expensive. Because of this, I considered both traditional feature selection, as well as cost sensitive feature selection that takes into account the cost of acquiring each feature. I compressed the learned model by estimating the contribution of different features and feature groups to the random forest model. Across the four groups of features I found that each group contained some important features, but the user reputation features representing the history of user contributions are most important, so I could not ignore this group of features. This motivated a focus on individual features to determine which specific features (instead of groups of features) contributed most significantly to the model. Using lasso, I found a minimum set of features whose classification performance is almost as good as the one with all 66 features.

Furthermore, I did cost sensitive feature selection to force the learning to prefer well-performing inexpensive features over more expensive ones that might yield only slight advantage in classification results. In combination, these techniques helped me train a compact, sparse vandalism detection model that scales well and performs well in
UGC systems can apply vandalism detection to help users understand which content may be problematic. Colorizing text is a common interface mechanism to indicate the trust level of any given submission. With my compact model, I know which features contribute most to the prediction of vandalism and could colorize or annotate the content to indicate the strength of the prediction.

Another variant of this technique is to help end-users detect and remove vandalism. Few user tools for vandalism removal provide any suggestion or prediction of that is or is not vandalism. This is not a complete oversight on the part of the tool designers. Some of the predictive models are difficult to maintain and the predictions cannot be computed in real-time, as the tool loads a specific contribution. The model that I develop could meet these requirements, but admittedly, I have not tested this in an actual vandal fighting tool. Instead, I have left that for some future work.

A third application of this technique is to use it as the basis for an awareness tool or event notification tool. Most wikis support some form of a watch list. When a user puts a specific wiki page on her watch list she is indicating interest in that page. When that page is edited or changed, the system sends email to the user about the edit. This is not a problem if a user is interested in a few pages, but if she were interested in many pages this might result in an unmanageable flood of email. An enhanced watch list mechanism could use my model, and a user specified vandalism threshold, to control notifications. That is, for some pages, a user might want to see every change (low vandalism threshold), while for others she might want to see only changes that cross a high threshold of predicted vandalism. This would allow a user to monitor a much broader span of pages while focusing her attention on pages or on activities that are of more interest to her.
Vandalism detection is a difficult problem. Technical approaches like mine are making progress. However, some user studies attempting to classify vandalism have illustrated that, even among users, there are types of vandalism that are open to interpretation [92, 114]. I used Amazon Mechanical Turk to begin re-labeling the PAN corpus using categories derived from [92] and [114]. I reported preliminary results based on binary classification of vandalism in each category. In future work I plan to explore the types of vandalism as a multi-label classification problem. My plan is to move beyond a standard binary classification (vandalism/legitimate) for each content edit, in order to explore where users agree or disagree on vandalism, in hopes of establishing some technical approaches that will help users understand what constitutes vandalism and how or why it occurs.

Vandalism will continue to be a challenge for all types of UGC systems. What constitutes vandalism will always be something “in the eye of the beholder”. As such, the community of users contributing to UGC systems will always be the final arbiters of what should stay and what should go.

Tools based on models like mine need to be timely and provide a clear and rational explanation for why a prediction is being made, so that the user can make the best decision possible with the information I provide.

8.2 Dissertation Highlights

Throughout this research I faced several challenges and learned many lessons. I hope to share them with other researchers in this area.
8.2.1 Make your system be general and scalable

In the early stages, I started to understand the importance of generality and scalability of my approach. Since 2006 when I started this research, Wikipedia has gone through numerous changes. The number of articles in English Wikipedia has increased from 1 million to more that 3.5 million in 2011. The number of languages it supports has also increased from 8 to 282. Thus, any application developed for Wikipedia needs to be capable of expanding to accommodate growth. It also needs to be general enough to be used in a multi-lingual platform like Wikipedia.

On a higher level, the application should be scalable and general enough to be applied to other UGCs. Since 2006, many new UGCs have been created. Twitter, Google+, and Kindle Forum are only a few of them. Other UGCs have also expanded significantly.

In most of these platforms spam/vandalism detection is a serious challenge. Matt Cutts, the head of Googles web spam group, offered cogent discussion on this issue in his talk on Aug 04, 2011 \(^1\). He mentioned that the generality of spam detection for social media data is very important. For a while Google had separate search tabs for wikis, Twitter (real-time), blogs, etc; and they used to treat these platforms differently. However, it is impossible to keep adding these tabs and any algorithm used in Google search should generally include the ones for spam detection. Interestingly, he discussed the issue of anonymous pages on the web and mentioned that the reputation of authors is a good predictor for spam detection. Therefore, any web mining approach that can measure reputation is highly to be valued.

The methods I use are not specific to wikis and can be applied to other UGCs. The new features I created, such as user reputation features in the form of DDSR and

\(^1\)http://www.youtube.com/watch?v=qlAydU6vBZo
usage of special characters in the text, were some of the most important features in the random forest model and can be easily extracted in other UGCs. Usage of special characters has been widely adopted as a criterion for spam detection in emails or blog comments, but user features, such as DDSR which measure the survivability of a user’s contribution, generally have not been used. This notion of survivability can be translated to other domains as well. For example, for Twitter we can see how often and how quickly a tweet gets re-tweeted. Similarly, in Facebook, we can look at patterns of sharing and propagation to measure survivability.

For the issue of scalability, from the very beginning of this study, I adopted a distributed approach. Recently, I have tried to map what I had done during this work into a MapReduce paradigm, as the most common distributed platform, in order to ensure that what I did is reproducible. Any task done in this work, like mining Wikipedia pages to extract token ownerships, feature extraction, learning binary classifiers, parameter tuning, etc can be done in a MapReduce paradigm. I used Amazon AWS clusters for all of these tasks and showed that the approach scales well.

8.2.2 Vandalism is fuzzy: look at your application

According to Wikipedia, vandalism is any addition, removal, or change of content in a deliberate attempt to compromise the integrity of Wikipedia. However, in many cases it is difficult to know whether a user’s changes or deletions have been intentional. In fact, smart vandals know how to make their edits look natural. So the border between vandalism detection and the filtering of low quality content, is not always clear.

This ambiguous border has caused researchers to consider low quality content as vandalism. For example, in [92], the authors consider nonsense or misinformation as vandalism. In addition, there are many cases in PAN 2010 corpus wherein I suspect
that the vandalism has been done deliberately: numerous edit error vandalism cases, for example.

Having “deliberate attempt as part of the definition of vandalism is very tricky. The same is true for web spam. After more than a decade, there is no clear-cut definition for web spam. I therefore feel that definitions for vandalism will always be fuzzy; but with the application of vandalism detection, we can adjust this definition. One good example is how Google treats spam. The ultimate goal is to improve the quality of ranking search results. So regardless of the user intent behind putting irrelevant links, they call it spam link and exclude it from the page rank calculation. Throughout this research, I used a similar approach. I considered low quality content as vandalism, regardless of the user intention.

To demonstrate how this definition affects the approach in solving the problem, let me consider one example. When modeling user reputation, I measured the survivability of the contributed content. Low quality was deleted by other users. Hence, I came up with very general features effective for detecting low quality content, including intentional vandalism. I showed the importance of these features. Only using DDSR results in AUC = 0.86. However, most of the features in other groups such as textual features are effective for detecting explicit and intentional vandalism.

8.2.3 Having a good feature set is the golden key

I somehow underestimated the feature extraction phase for vandalism detection. I tried to combine the entire existing feature sets in the literature and I also added numerous other valuable features. However, after doing vandalism categorization, using Amazon’s Mechanical Turk, I had a much deeper understanding of vandalism. I found out that for some categories, such as edit error or comment vandalism, I could have
more features. So I should have studied these vandalism categories before the feature extraction phase as humans. Once we have learned how to detect different types of vandalism, we can translate our perceptions to a set of mechanized, understandable features.

I also spent a lot of time and resources on learning the binary classifiers, trying to improve the classification performance. I tried different binary classifiers with large scale tuning. In addition, I tried different methods of handling missing data and class imbalance problems. Most of these methods resulted in some insignificant improvements in the classification performance. However, in the last steps of the work, I added one feature to the feature set that resulted in a significant gain in the classification performance, from 0.9553 to 0.9566.

I suspect that it is wiser to spend more time and resources on the feature extraction part rather than training complex binary classifiers. A very good example is spam detection in emails. After more than a decade, it is widely known that Naive Bayes classifiers are good enough to handle this problem. In this case, I did not have independent features, so I could not use Naive Bayes; but as I have already demonstrated, simple random forests have outperformed all other classifiers. Random forests are much simpler than other classifiers that require extensive tuning.
Bibliography


