Detecting Vandalism in Wikipedia

Xabriel J Collazo Mojica

3/2/2022

Introduction

Wikipedia is a free online encyclopedia that is written and maintained by volunteers. Anyone with internet access can edit Wikipedia. This model has enabled Wikipedia to grow steadily since its inception in 2001 to the current 58 million articles in 325 languages. As of November 2020, Wikipedia is edited 17 million times per month (1.9 edits per second) [12].

Most of the edits are bona fide contributions. However, about 7% of them are vandalism. Wikipedia defines vandalism as "any change or edit that manipulates content in a way that deliberately compromises Wikipedia's integrity" [13]. Vandalism examples include adding out of context profanity, nonsense, removing content without a reason, and adding plausible but false content.

Wikipedia vandalism is removed with manual and automated approaches. In manual approaches, vandalism is deleted by the same community of contributors. In severe cases, where vandalism occurs repeatedly for a particular entry, administrators may enable edit restrictions. Both of these manual approaches slow down the rate of quality contributions since time has to be diverted to clean up existing content, or bona fide contributions are temporally not allowed because of the restrictions.

In automated approaches, software bots have been implemented to detect and correct vandalism. The first iteration of bots used heuristics and regular expressions to detect potential vandalism. The second and current iteration utilizes more sophisticated algorithms. As an example, the "ClueBot NG" bot uses Naive Bayes and Neural Networks to detect vandalism in the English Wikipedia. It catches about 40% of vandalism with its current setting of maximum 0.1% false positive rate [14]. This is the current state of the work.

There has been considerate research done on improving Wikipedia vandalism detection [5], [6], [4], [2]. One of the most interesting approaches is that of [6], in which a competition was held to encourage detection contributions to the state of the art. We utilize the same dataset as in [6] to implement a binary classifier that can be compared to the top submission of their competition.

The dataset is the "PAN Wikipedia Vandalism Corpus 2010." It consists of 32,452 edits on 28,468 English Wikipedia articles, among which 2,391 vandalism edits have been identified [5]. It reflects the same rate of vandalism as Wikipedia.

The dataset consists of multiple files with metadata about the edits such as a unique identifier, the author of the edit, and whether the edit is considered vandalism. It also includes the content of the edited article before and after the edit, and thus we can use existing textual difference algorithms to find the additions and deletions. We discuss the dataset and transformations in more details in the Methods and Analysis section.

We define the problem as follows: given a set of edits of Wikipedia articles, we want to separate the edits we believe to be vandalism, from those we believe not to be. Given that most edits are bona fide, this is an imbalanced binary classification problem. Existing literature suggest that an appropriate method to measure performance is with a "Precision-recall curve" (PR Curve) [6], [9]. This method allows us to visually compare the trade-off between these metrics, and by calculating the area under the curve of this method (PR-AUC), we can summarize the performance to a single number. This number can then be used to choose the best parameters when tuning an individual model, and to compare between models.

We find that by implementing a set of 24 features that rely on the provided dataset without any other external source, we achieve a PR-AUC of 0.6608231 with a Random Forest algorithm trained with 1000 trees and 3 random features at each split point. This result is on par with the best approach discussed in [6] which yields 0.6628893.

Methods and Analysis

PAN Wikipedia Vandalism Corpus 2010

The PAN Wikipedia Vandalism Corpus 2010 consists of 32,452 edits on 28,468 English Wikipedia articles, among which 2,391 vandalism edits have been identified [5]. It reflects the same rate of vandalism as Wikipedia at 7.38%. There are two files and one directory included in the dataset that are relevant to our work. The first file, `edits.csv`, contains comma separated fields, with each line containing the following:

Field	Description
editid	A synthetic unique edit id to easily join this file with others.
editor	The username for this edit, or an IP address if anonymous.
oldrevisionid	A Wikipedia unique id for the revision of the article before the edit.
newrevisionid	A Wikipedia unique id for the revision of the article after the edit.
diffurl	a URL pointing to Wikipedia's visual interface for comparing old and new revision.
edittime	Timestamp for the edit in ISO 8601 UTC format.
editcomment	The comment, if any, that the user left when making the edit.
articleid	A Wikipedia unique id of the article that this edit modifies.
articletitle	The title of the edited article.

The second file, `gold-annotations.csv`, contains the following:

Field	Description
editid	References the editid from `edits.csv`.
goldclass	Classification of the edit as vandalism or regular.
annotators	Number of annotators that concur with the classification
totalannotators	Total number of annotators who reviewed and classified the edit.

Finally, there is a directory called `article-revisions/` which [5] describes as: "A list of directories `part1/` through `part65/`. Each part directory contains up to 1000 text files, each of which being the revision of a Wikipedia article. The name of a file is its revision identifier, which is referenced in the `edits.csv` file. The contents of each text file is the plain wikitext of the revision."

Data cleaning

For the rest of this report we use the R programming language and supporting libraries to transform the dataset [11]. A working knowledge of programming and regular expressions is assumed.

The clean up steps for `edits.csv` and `gold-annotations.csv` is trivial: we need only to read and join them by `editid`. We keep the output in an `edits` R data frame object.

Processing `article-revisions/` is more interesting. Our end goal is to calculate what is commonly known as a `diff`; the textual difference between two files. We first define a function `git_diff()` that leverages the built-in diff algorithm from the git version control system[1]. The key argument to this call is

[1] We considered multiple approaches for calculating the diff, including using portable code from the `diffobj` R package. However, we found no elegant way to transform the diff into a format amenable to our purposes other than with `git diff`. If the reader is attempting to reproduce our code, we recommend the use of a Unix-like system like Linux or MacOS.

--word-diff=porcelain, which creates an output that is in a line-based format easily consumed by scripts [1].

```r
git_diff <- Vectorize(function(old, new) {
  system2(
    command = "git",
    args = c(
      "diff",
      "--no-prefix",
      "--no-index",
      "--word-diff=porcelain",
      "--unified=0",
      old,
      new
    ),
    stdout = TRUE
  )
})
```

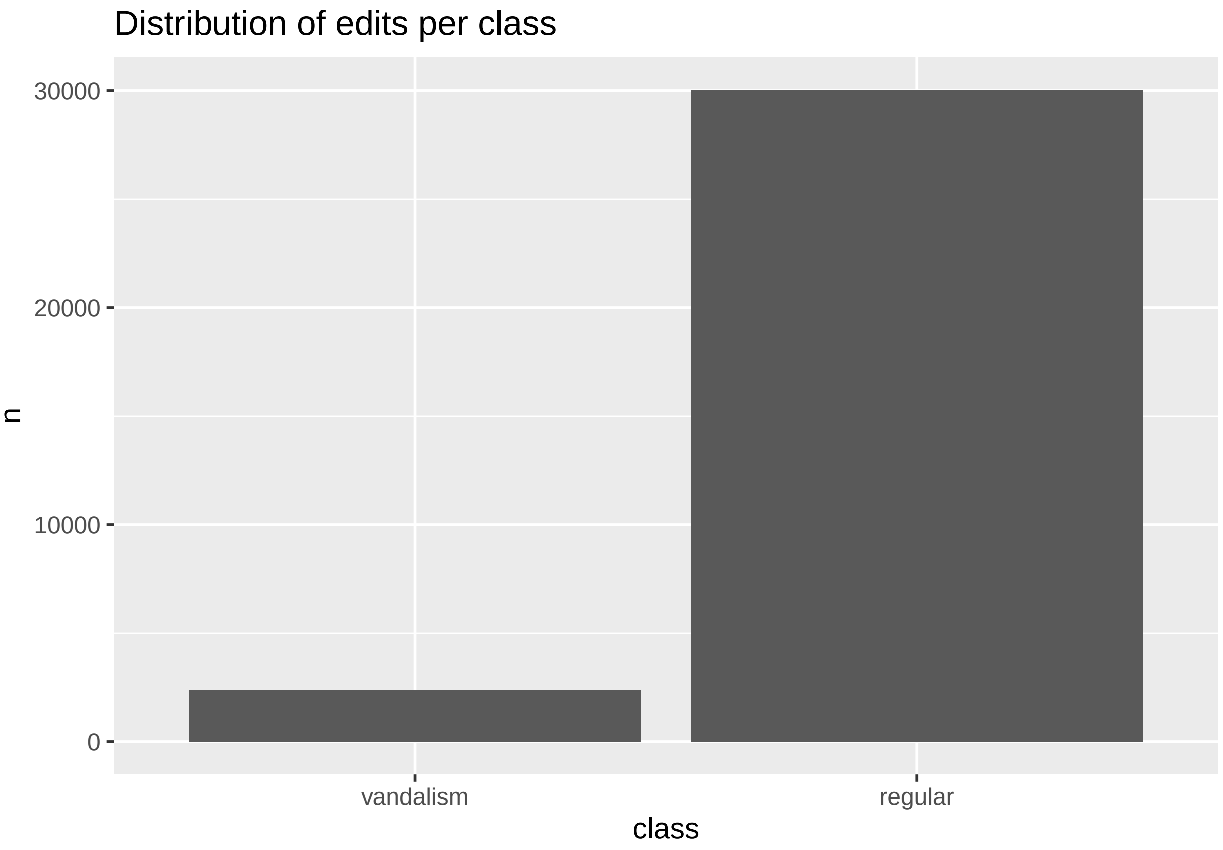
With this function defined, we now walk the article-revisions/ folder collecting metadata about each revision, and then join this data with edits. We then mutate the data frame further by calling git_diff() on each pair of 'old' and 'new' revisions, and then we separate the diff output into additions (lines that start with an +) and deletions (-). Finally, we join the difference calculations into edits to consolidate into one object.

```r
parentPath <- "data/pan-wikipedia-vandalism-corpus-2010/"

# recursively find files in the article-revisions/ directory
revisionPaths <-
  list.files(
    path = paste(parentPath, "article-revisions", sep = ""),
    full.names = TRUE,
    recursive = TRUE
  )

# create a data frame with revision information
revisions <- map_dfr(revisionPaths, function(path) {
  list(
    revisionid = as.integer(str_remove(basename(path), ".txt")),
    revisionpath = path,
    revisionsize = file.size(path)
  )
})

# calculate diff for each edit
diffs <- edits %>%
  select(editid, oldrevisionid, newrevisionid) %>%
  left_join(
    revisions %>%
      select(revisionid, revisionpath, revisionsize) %>%
      rename(oldrevisionpath = revisionpath, oldrevisionsize = revisionsize),
    by = c("oldrevisionid" = "revisionid"),
    copy = TRUE
  ) %>%
  left_join(
    revisions %>%
      select(revisionid, revisionpath, revisionsize) %>%
      rename(newrevisionpath = revisionpath, newrevisionsize = revisionsize),
    by = c("newrevisionid" = "revisionid"),
    copy = TRUE
  ) %>%
  # diff is a list of chars
  mutate(diff = git_diff(oldrevisionpath, newrevisionpath)) %>%
  select(editid,
         oldrevisionid,
         newrevisionid,
         oldrevisionsize,
         newrevisionsize,
         diff) %>%
  # additions are diff lines that start with '+'
  mutate(additions =
           lapply(diff, function(d) {
             s = d[str_starts(d, fixed("+")) & !str_starts(d, fixed("+++"))]
             str_sub(s, start = 2)
           })) %>%
  # deletions are diff lines that start with '-'
  mutate(deletions =
           lapply(diff, function(d) {
             s = d[str_starts(d, fixed("-")) & !str_starts(d, fixed("---"))]
             str_sub(s, start = 2)
           }))

edits <- edits %>%
  select(-diffurl) %>%
  left_join(diffs %>% select(-oldrevisionid,-newrevisionid),
            by = c("editid"))
```

Data exploration

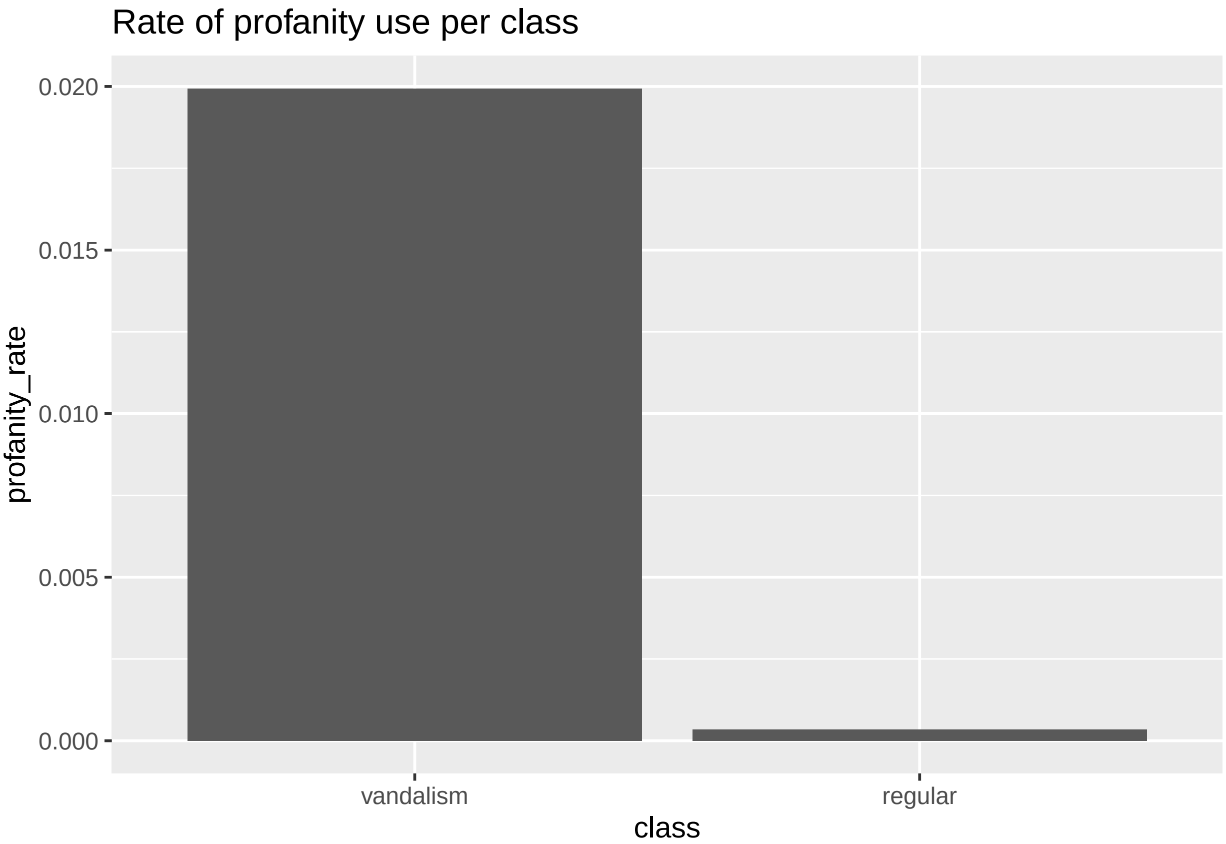
As alluded previously, the dataset is highly imbalanced with the `vandalism` class being 7.38% of the data:

```
edits %>%
  collect() %>%
  group_by(class) %>%
  summarise(n = n()) %>%
  ggplot(aes(x=class, y= n)) +
  geom_bar(stat="identity") +
  ggtitle("Distribution of edits per class")
```



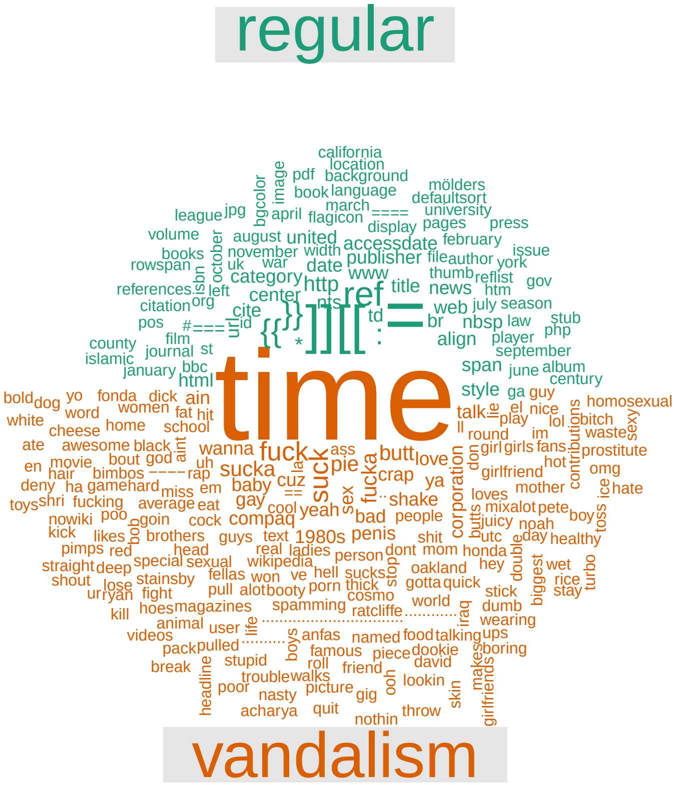
However, the `vandalism` class does show trends that can drive the classification. As an example, most of the profanities are found in the `vandalism` class:

```
# load list of profanities
profanities <- read_lines(file = "data/en/profanities.txt")
# a regex that is aware of wikisyntax
wiki_regex <- "----|:{1,6}|\\*{1,4}|#{1,4}|={1,5}|\\{\\{|\\}\\}|\\[\\[|]]|\\w+"
# setup for parallelism
multidplyr::cluster_copy(cluster, 'bin_search')
multidplyr::cluster_copy(cluster, 'profanities')

# extract words from the diff additions, and check whether each one of them
# against a profanity word list. Then calculate the profanity rate for each class.
words_by_class <- edits %>%
  collect() %>%
  select(class, additions) %>%
  mutate(
    additions_as_string = map_chr(additions, str_c, collapse = "\n"),
    word_list = map(str_match_all(additions_as_string, wiki_regex), as.vector),
    word_list_lower = map(word_list, str_to_lower),
  ) %>%
  select(class, word_list_lower) %>%
  unnest_longer(col = word_list_lower, values_to = "word") %>%
  partition(cluster) %>%
  mutate(word = replace_na(word, ""),
         is_profanity = map_lgl(word, function(w) {
           bin_search(profanities, w) > 0
         })) %>%
  collect() %>%
  group_by(class)

words_by_class %>%
  summarise(profanity_rate = sum(is_profanity) / n()) %>%
  ggplot(aes(x = class, y = profanity_rate)) +
  geom_bar(stat="identity") +
  ggtitle("Rate of profanity use per class")
```



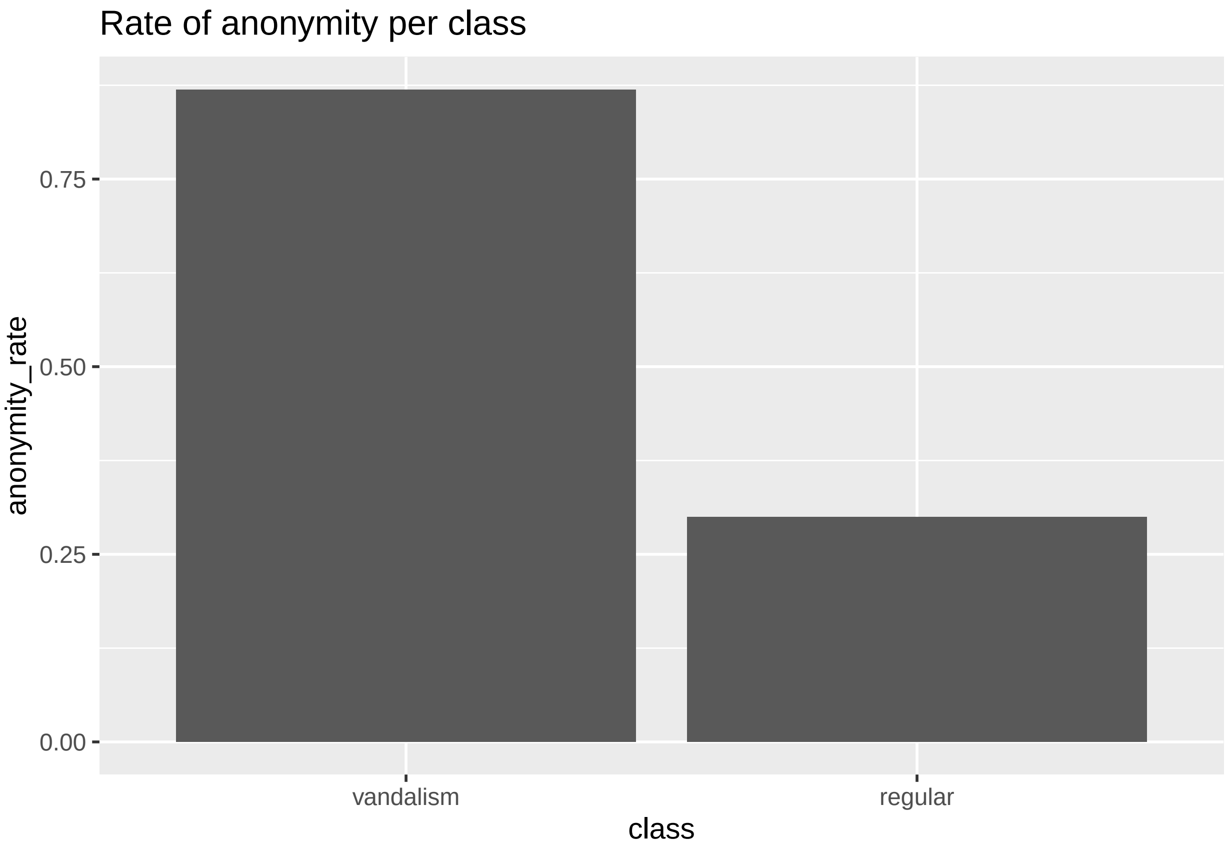
Similarly, vandals tend to use words that differs from bona fide edits. In the comparison wordcloud below, we can see that vandals tend to use profanities and contractions, and notably the word 'time,' while bona fide edits tend to use wikisyntax and URLs.

```
word_counts <- words_by_class %>%
  anti_join(stop_words) %>%
  filter(!str_detect(word, "^\\d+$")) %>% # no numbers
  group_by(class, word) %>%
  summarise(n = n()) %>%
  arrange(desc(n)) %>%
  pivot_wider(names_from = class, values_from = n, values_fill = 0L) %>%
  # normalize vandalism words hits since it is 7.38%
  mutate(vandalism = as.integer(vandalism * 1/.0738))

word_counts_matrix <- as.matrix(word_counts[,-1])
rownames(word_counts_matrix) <- word_counts$word

comparison.cloud(
  word_counts_matrix,
  random.order=FALSE,
  title.size = 2L,
  match.colors = TRUE
)
```



Additionally, most of the vandals are anonymous:

```
# a regex that matches IP addresses
ip_address_regex <- "\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}"

edits %>%
  collect() %>%
  select(editor, class) %>%
  mutate(is_anonymous = str_detect(editor, ip_address_regex)) %>%
  group_by(class) %>%
  summarise(anonymity_rate = sum(is_anonymous) / n()) %>%
  ggplot(aes(x = class, y = anonymity_rate)) +
  geom_bar(stat="identity") +
  ggtitle("Rate of anonymity per class")
```



These and other signals can then be used to construct a feature set that tries to separate the vandalism from the bona fide contributions. Note however that not all profanity use is vandalism, and not all anonymous contributors are vandals. Similarly, not all contraction use is illegitimate. Thus our main insight is that since the signals seem to be blurry, we need to be able to evaluate our solution in a way that penalizes classifying bona fide contributions as vandalism.

Evaluating performance

There are multiple ways to evaluate the performance of a learning algorithm. Metrics are typically derived from the confusion matrix. That is, from the four possible predicted conditions: true positive (TP), true negative (TN), false positive (FP), or false negative (TP). Below we discuss our chosen metrics that allow us to compare our solution against both research classifiers as well as the current system used by Wikipedia.

Metric for comparison againt top classifiers from [6] For imbalanced binary classification, a metric like $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ can be misleading since we would get a high accuracy even if we always predict the majority class. In our case where the `vandalism` class is 7.38% of the dataset, always predicting the majority class would yield 92.62% accuracy.

Additionally, we need a metric that penalizes FP hits, since labeling bona fide contributions as `vandalism` is a particularly bad outcome for a system as Wikipedia that depends on volunteer contributions. Existing literature suggest that an appropriate metric for this situation is the "Precision-recall curve" (PR Curve). This curve allows us to visually compare the trade-off between precision and recall, and provides visual separation when comparing alternative classifiers. By calculating the area under the curve (PR-AUC), we can summarize the performance to a single number [6], [9].

PR Curves are calculated as follows: 1) Implement a binary classification algorithm. 2) Use the algorithm to predict the probability of the 'positive' class (in our case `vandalism`). 3) Now calculate the confusion matrix of the algorithm while varying the threshold τ of binary classification between [0,1] with respect to the positive class probability. 4) Use the confusion matrix at each threshold τ to calculate the Precision and Recall, and plot the outcome as x and y, respectively.

Metric for comparison against ClueBot NG [14] We also compare our classifier to the current vandalism bot in the English Wikipedia. Only two performance metrics seem to be available: 40% TP-rate at fixed 0.1% FP-rate, and 55% TP-rate at fixed 0.25% FP-rate [14][2].

These two data points can be used as reference points in a Receiver Operating Characteristic Curve (ROC Curve). A ROC Curve is constructed similarly to a PR curve, but instead we plot the TP rate against the FP rate. This is the only comparison that we can do against the limited data available from ClueBot NG's performance.

Validation and test data In this work, the validation set will be 50% of source data. We do this to be comparable to the classifiers discussed in [6], where a 50% validation set was used as well. Given half of the data will be used for validation, we opt to do the training using 10-fold cross validation, instead of further partitioning the remaining data into a discrete test set.

Feature engineering

The following set of features have been implemented from descriptions of the work of [6] and [4]. They were extracted from the dataset without aid of external sources of information. Unless otherwise noted, these features were derived from the additions of the diff. We considered features from the deletions, but they either yielded insignificant gains or hurt performance.

Category	Feature	Description
Character-level	`upper_to_lower_ratio`	The ratio of uppercase characters (`[A-Z]`) to lowercase (`[a-z]`).
Character-level	`upper_to_all_ratio`	The ratio of uppercase characters (`[A-Z]`) to all characters (`.`).
Character-level	`digits_ratio`	The ratio of digits (`\\d`) to all characters (`.`).
Character-level	`special_chars_ratio`	The ratio of special characters (`[^A-Za-z0-9]`) to all characters (`.`).
Character-level	`char_diversity`	The character length of all additions to the (1 / number of different characters).
Character-level	`compression_ratio`	The ratio of the length of all additions to the length of the compressed version of all additions. We used the `gzip` coder for its speed.
Word-level	`profanity_count`	How many inserted words are considered to be profanities.
Word-level	`pronoun_count`	How many inserted words are pronouns. Ex: He, she, they.

[2] After perusing ClueBot NG's Request for approval at https://en.wikipedia.org/wiki/Wikipedia:Bots/Requests_for_approval/ClueBot_NG, it seems like at some point there was a 'trial report,' that is, a detailed document with perhaps a full ROC Curve, but this report has been lost as it was kept in a personal server rather than as part of Wikipedia. We asked for help to find this document at https://web.libera.chat/?channel=#wikipedia-en-help, but as of the time of publishing, the report has not been found.

Category	Feature	Description
Word-level	`superlative_count`	How many inserted words are superlatives. Ex: biggest, greatest.
Word-level	`contraction_count`	How many inserted words are contractions. Ex: methinks, won't).
Word-level	`wikisyntax_count`	How many inserted words are considered WikiSyntax Ex: "[[," table.
Word-level	`common_vandalism_count`	The top 200 words used by vandals calculated from the train set.
Word-level	`common_regular_count`	The top 200 words used by non-vandals calculated from the train set.
Word-level	`longest_word`	The length of the longest word inserted.
Comment-level	`comment_exists`	Whether the editor included a comment or not.
Comment-level	`comment_length`	The length of the comment if included, otherwise 0.
Comment-level	`comment_is_revert`	Whether the comment content suggests the edit is a revert of a previous edit.
Comment-level	`comment_is_bot`	Whether the comment content suggests the edit was made by a bot.
Comment-level	`comment_has_profanity`	Whether the comment content includes any word considered a profanity.
Size statistic	`size_delta`	The difference between the byte length of the new and old revisions.
Size statistic	`size_ratio`	The ratio between the byte length of the new and old revisions.
Size statistic	`num_additions`	The number of separate additions as derived by the diff algorithm.
Size statistic	`num_deletions`	The number of separate deletions as derived by the diff algorithm.
Editor reputation	`is_anonymous`	Whether the editor is registered on Wikipedia or not.

Machine Learning Algorithm

In this work we considered three different algorithms: K-nearest Neighbors (kNN) [8], [10], Neural Networks (NN) [7] and Random Forest (RF) [3]. kNN was an attractive baseline algorithm because of its fast training time. NN was interesting to consider since there is established use in Wikipedia for their ClueBot NG bot [14]. Random forests have also been successfully used for Wikipedia vandalism [6], [4].

Results

Training results

kNN We train kNN with $k = 3, 5, 7, \ldots, 21$. Normalizing the features to the range $[0, 1]$ helps the algorithm significantly, from a best training PR-AUC of 0.3503731 at $k = 15$ without normalization, to 0.3904509 at $k = 19$. However, the greatest improvement is achieved when we normalize and weight closer neighbors more. In a weighted kNN with same k range, we obtain a best training PR-AUC of 0.5253941 at $k = 21$. The weighted kNN algorithm allows us to specify the distance metric. We considered Manhattan and Euclidean distances, finding that the Manhattan distance achieves marginally better performance.

Neural networks We chose a NN that requires a *size* and *decay* parameter. The *size* relates to the number of neurons in the hidden layer of the network. It is recommended to use as many neurons as necessary, but not more, since extra neurons may overtrain the network. Similarly, the *decay* parameter is a form of regularization that penalizes the NN optimization function as to help not over train. We consider $size = 1, 2, 3, 4, 5, 10, 15, 20, 25$ and $decay = 0, 0.1, 0.2, \ldots, 1$. We tried regularizing the features to [0,1] but that only hurt performance. We find a best training PR-AUC of 0.6384951 with $size = 25$ and $decay = 0.9$.

Random Forest The RF algorithm can be tuned with two variables: *ntree*, which is the number of trees in the forest, and *mtry*, the number of randomly chosen features at each split point. We train with $ntree = 150, 500, 1000$, and $mtry = 1, 2, 3, 4, 5, 10, 25, 50, 100$. All features were regularized to [0,1] as this yielded a minor improvement. We find a best training PR-AUC of 0.6528348 with $ntree = 1000$ and $mtry = 3$.

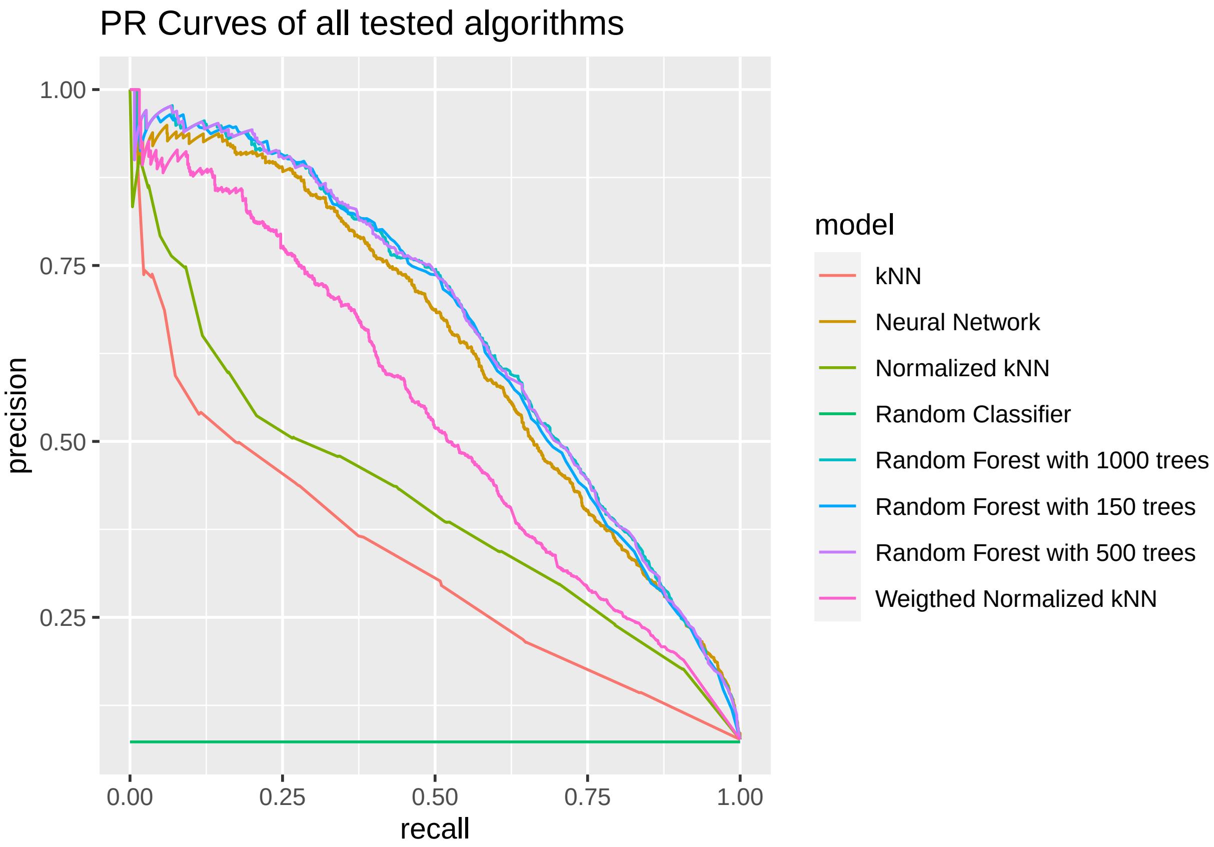
Validation results

Below we can find a table summarizing the results for PR-AUC and ROC-AUC metrics when using the validation set. We find that the best algorithm, as with the training set, is the Random Forest with 1000 trees.

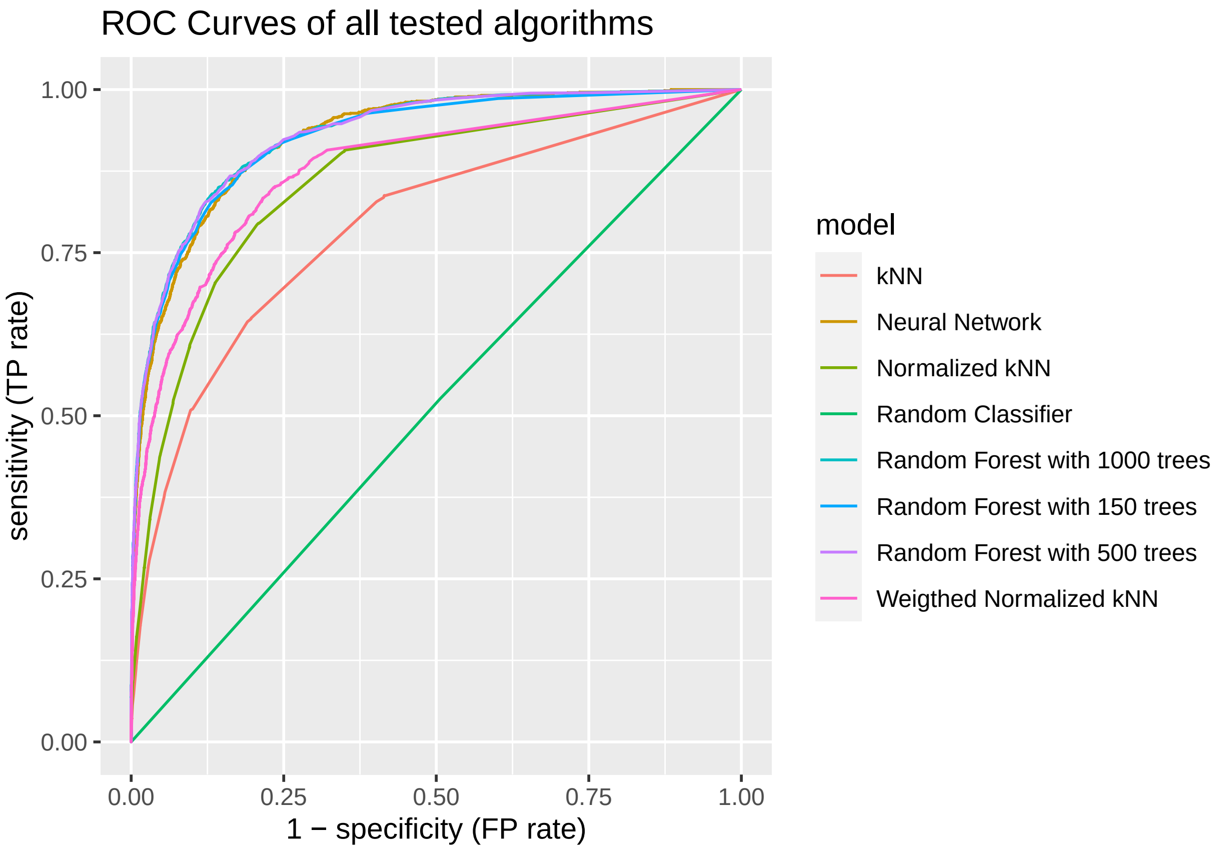
Table 4: PR-AUC and ROC-AUC of all tested algorithms

model	PR-AUC	ROC-AUC
kNN	0.3299779	0.7897331
Normalized kNN	0.4139975	0.8591455
Weigthed Normalized kNN	0.5338900	0.8779897
Neural Network	0.6348329	0.9275763
Random Forest with 150 trees	0.6551328	0.9261026
Random Forest with 500 trees	0.6603834	0.9304145
Random Forest with 1000 trees	0.6608231	0.9306720
Random Classifier	0.3207735	0.5098613

The PR Curves below show that the three Random Forest classifiers are clearly in the lead over the whole space, while the Neural Network classifier is a close 4th place. The three kNN classifiers perform the worst.



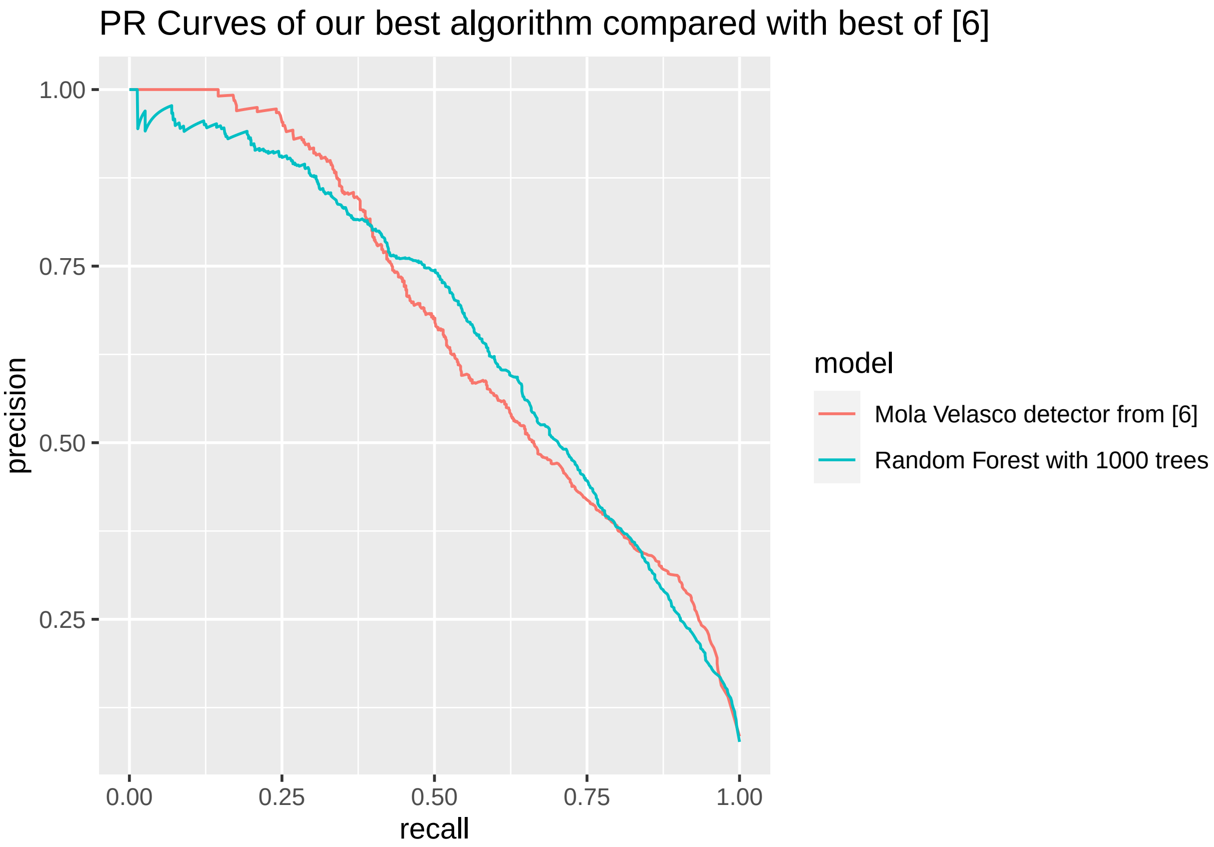
In ROC space, we can appreciate the same ranking as in PR space. Note how it is significantly more difficult to separate the algorithms, thus agreeing with [9] in that a PR curve is more informative when comparing imbalanced binary classifiers.



Comparison against top classifier from [6][3]

In this comparison, we can see that our solution performs similarly as the best from [6]. In PR Curve space, we perform slightly better on recall values between 0.4 and 0.75, but otherwise their solution has a slight advan-

[3] We would like to acknowledge the help from Dr. Martin Potthast in making the classification runs from [6] available at https://drive.google.com/file/d/1Q1fSGZWU3rGopnyCGxKpqHn507rm0CKt/view?usp=sharing.



tage.

Table 5: PR-AUC and ROC-AUC of best algorithm and best from [6]

model	PR-AUC	ROC-AUC
Random Forest with 1000 trees	0.6608231	0.9306720
Mola Velasco detector from [6]	0.6628893	0.9243112

Comparison against ClueBot NG

Our algorithm with the best PR-AUC, a Random Forest with 1000 trees, has an TP-rate of ~42.5% at 0.01% FP-rate, and ~57.5% TP-rate at 0.025% FP-rate. This compares favorably against ClueBot NG, resulting in a ~5% improvement overall. The best algorithm from [6] has a TP-rate of ~40% at 0.01% FP-rate, which is as good as ClueBot NG, and a TP-rate of ~51.5% at 0.025%, which is a ~3.6% decrease from ClueBot NG. Below we present a ROC Curve that has been zoomed in to detail the only two known data points from ClueBot NG [14].

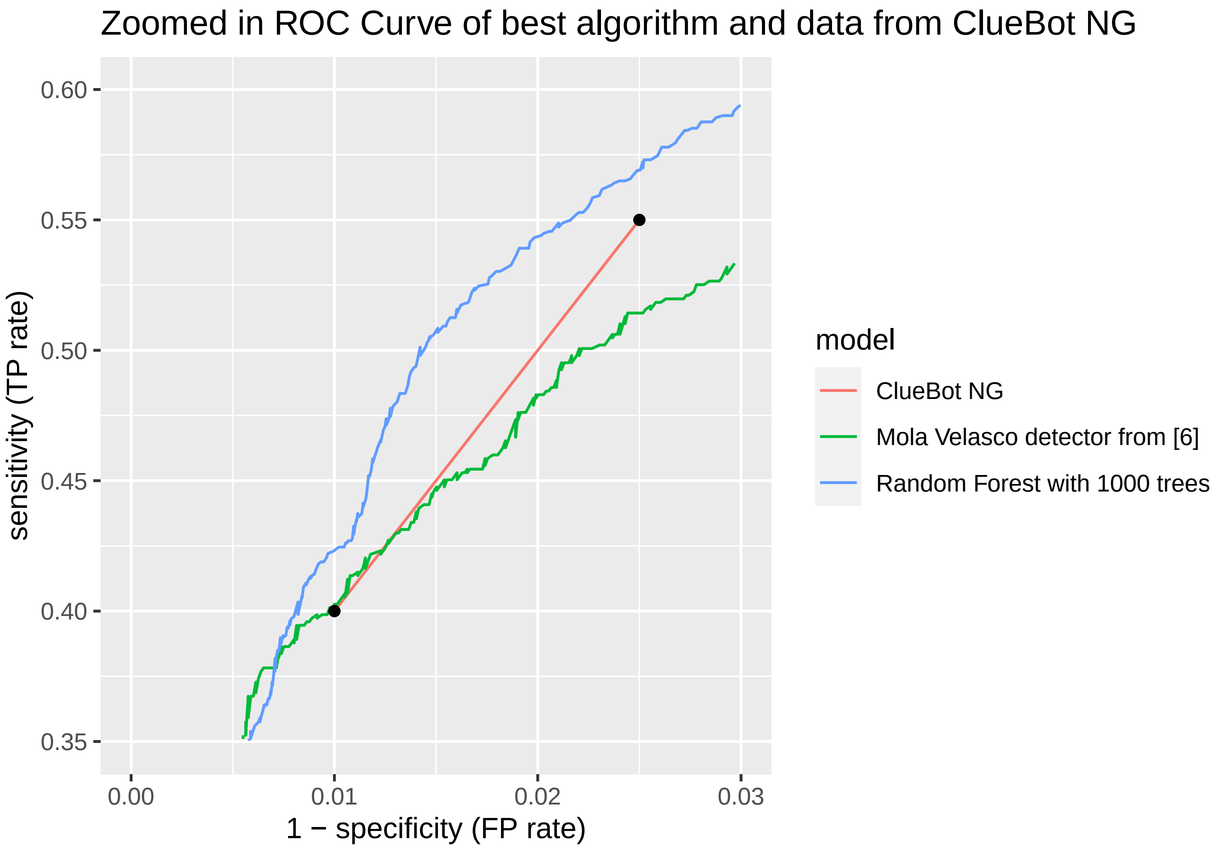


Table 6: Practical comparison points of best algorithms and ClueBot NG

model	TP-rate @ 0.01% FP-rate	TP-rate @ 0.025% FP-rate	Average Improvement % from ClueBot NG
Random Forest with 1000 trees	42.5	57.5	5.263158
ClueBot NG	40.0	55.0	0.000000
Mola Velasco from [6]	40.0	51.5	-3.684210

Conclusion

In this report we have discussed the problem of detecting vandalism in Wikipedia, an online encyclopedia that anyone can edit. We utilize the PAN Wikipedia 2010 Vandalism corpus to derive a set of 24 features. We discuss, train, and validate a set of binary classification algorithms. We find that Random Forests yield the best performance both from a PR-AUC, as well as a ROC-AUC perspective. Specifically, a Random Forest with 1000 trees and 3 random features at each split point yields the best performance.

When we compare our best classifier with the best from [6], we find similar PR-AUC performance. When we compare it against [14], Wikipedia's current production classifier, we find our solution has an average ~5% advantage at a FP-rates of 0.01% and 0.025%. In other words, out of every 10,000 edits that are true vandalism, we can detect 4,250, while out of every 10,000 edits that are true bona fide, we incorrectly label 1. The current system detects 4,000 and 1, respectively.

We focused on maximizing the PR-AUC metric for our work to be comparable to [6]. We find that this metric has strong research merit, but it doesn't translate to the core Wikipedia vandalism problem: they need a sufficiently low FP-rate to be able to justify the use of a vandalism bot. That is, their main consideration is that the classifier should not deter bona fide contributions. In future work, we will focus on improving the TP-rate with the constraint that the FP-rate <= 0.01%, instead of trying to maximize PR-AUC.

It is reasonable to expect that a production solution for Wikipedia vandalism should strive to be easy to interpret. A user that makes a contribution that is classified as `vandalism` should understand why. We believe a comprehensive report that explains the methods and includes the source code like ours helps with this interpretability issue. However, we would like to explore if simpler, self-describing algorithms such as Decision Trees could achieve reasonable FP-rates while also being less opaque for lay users.

Finally, it has been argued that the use of features such as whether a user is anonymous, or whether they use contractions, can be an inequitable treatment of contributors [2]. We agree that these features could lead to biased FP hits. It would be interesting then to explore in future work how to develop anti-vandalism systems that take this critique into account.

References

[1] Git contributors. 2022. Git - git-diff documentation. Retrieved from https://git-scm.com/docs/git-diff#Documentation/git-diff.txt-porcelain

[2] Paul Laat. 2015. The use of software tools and autonomous bots against vandalism: Eroding wikipedia's moral order?. *Ethics & Information Technology* 17, 3 (2015), 175–188. Retrieved from https://search-ebscohost-com.research.aadl.org/login.aspx?direct=true&db=aci&AN=110319992&site=ehost-live

[3] Fortran original by Leo Breiman, R port by Andy Liaw Adele Cutler, and Matthew Wiener. 2022. *randomForest: Breiman and cutler's random forests for classification and regression.* Retrieved from https://cran.r-project.org/package=randomForest

[4] Santiago M. Mola-Velasco. 2012. Wikipedia vandalism detection through machine learning: Feature review and new proposals: Lab report for PAN at CLEF 2010. Retrieved from http://arxiv.org/abs/1210.5560

[5] Martin Potthast. 2010. Crowdsourcing a Wikipedia Vandalism Corpus. In *33rd international ACM conference on research and development in information retrieval (SIGIR 2010)*, ACM, 789–790. DOI:https://doi.org/10.1145/1835449.1835617

[6] Martin Potthast, Benno Stein, and Teresa Holfeld. 2010. Overview of the 1st International Competition on Wikipedia Vandalism Detection. In *Working notes papers of the CLEF 2010 evaluation labs* (Lecture notes in computer science). Retrieved from http://ceur-ws.org/Vol-1176/

[7] Brian Ripley and William Venables. 2022. *Nnet: Feed-forward neural networks and multinomial log-linear models.* Retrieved from https://cran.r-project.org/package=nnet

[8] Brian Ripley and William Venables. 2022. *Class: Functions for classification.* Retrieved from https://cran.r-project.org/package=class

[9] Takaya Saito and Marc Rehmsmeier. 2015. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PloS one* 10, 3 (March 2015), e0118432–e0118432. DOI:https://doi.org/10.1371/journal.pone.0118432

[10] Klaus Schliep and Klaus Hechenbichler. 2016. *Kknn: Weighted k-nearest neighbors.* Retrieved from https://CRAN.R-project.org/package=kknn

[11] R Core Team. 2021. *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. Retrieved from https://www.R-project.org/

[12] Wikipedia contributors. 2022. Wikipedia — Wikipedia, the free encyclopedia. Retrieved from https://en.wikipedia.org/w/index.php?title=Wikipedia&oldid=1071043064

[13] Wikipedia contributors. 2022. Vandalism on wikipedia — Wikipedia, the free encyclopedia. Retrieved from https://en.wikipedia.org/w/index.php?title=Vandalism_on_Wikipedia&oldid=1069412446

[14] Wikipedia contributors. 2022. User:ClueBot NG — Wikipedia, the free encyclopedia. Retrieved from https://en.wikipedia.org/w/index.php?title=User:ClueBot_NG&oldid=391868393