GitHub Issue Label Clustering by Weighted Overlap Coefficient

by

Yunlong Li
B.Eng., Shaanxi University of Science and Technology, 2014

A Master's Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

GitHub Issue Label Clustering by Weighted Overlap Coefficient

by

Yunlong Li
B.Eng., Shaanxi University of Science and Technology, 2014

Supervisory Committee

Dr. Daniela Damian, Supervisor
(Department of Computer Science)

Dr. Kelly Blincoe, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Daniela Damian, Supervisor
(Department of Computer Science)

Dr. Kelly Blincoe, Departmental Member
(Department of Computer Science)

ABSTRACT

GitHub labels are designed for helping people to classify and recognize different issues. When naming a label, people may use different word formats (e.g., bug, Bug, bugs, etc.) to express the same meaning. Therefore, managing the issue labels in GitHub becomes a challenging task. Clustering the morphological synonym labels will make it easier for management of the issues and complete some data preprocessing work for the automatic labeling research. String similarity calculation is the key part of the clustering algorithm. In this project, a weighted overlap coefficient method is proposed as a string similarity measure for clustering the labels. The most frequently used 200 labels are selected as the experiment data for analysis. The preliminary working results show that the new method does improve the original overlap coefficient by producing a 4.43% higher F-Measure and 92.42% of all the experiment labels have been correctly clustered.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Dr. Daniela Damian, for mentoring, support, encouragement, and patience.

My parents, for their endless love and full support.

Ali Dehghan, for raising the project idea and supporting me in the initial stage of the project.

Dr. Kelly Blincoe and Dr. Kui Wu, for serving in my oral defense committee.

DEDICATION

I dedicate this report to my loving parents

Chapter 1

Introduction

Nowadays, numerous companies and people are using GitHub[1] to control open source projects. With the scale of projects getting larger and larger, managing the project issues will become more and more difficult. In order to solve this problem, GitHub allows users to use issue labels (e.g., "enhancement", "bug"). By using the labels, people can identify issues or filter out certain kind of issues quickly.

However, when users name the GitHub labels, they can choose the label words freely. Although people want to express the same meaning, they may choose different words to name the labels due to differences of social and culture backgrounds. Taking "High priority" as an example, there are lots of similar labels to express the same meaning, such as "highpriority", "high_priority", "high priority", "high-priority", "priority:high", "priority high" and "priority-high". Another reason for different word formats is that some words may be misspelled. For instance, "enhancement" is misspelled as "enhancment". The different formats also come from the letter cases (upper vs. lower), plurals, tenses and so on. The reasons above lead to the problem that there are many labels existing in GitHub with similar meaning.

This project aims to cluster the morphological synonym GitHub labels and investigate the string similarity metrics as well as make some improvements.

1.1 GitHub and GitHub labels

GitHub is an distributed version control system. A version control system can keep the history of a collection of files and help people work collaboratively on a project

[1] https://github.com/

because big projects cannot always be done by a single person. When people work together, they may change the same file. The version control system[2] can notify programmers about the conflict in order to control the modification of the code. It can also store and manage the "commit history" to help people track the code. GitHub is a platform to manage the open source software projects. The name is GitHub because it only supports the Git format. Other than the version control function, GitHub also provides task management and an issues tracking system[3].

The issue labels[4] in GitHub can help people recognize the issues category, priority, difficulty or any other information. As the number of issues increases, managing them will become a big challenge. Labels can help classify and filter the issues to make the management work much more efficiently. GitHub has its own issue tracking system and label management system. As we can see from Figure 1.1 of the GitHub label management system, there are seven pre-defined labels[5] (i.e., "bug", "duplicate", "enhancement", "help wanted", "invalid", "question", "wontfix"). Apart from these, users can also add, edit and delete the labels here. For example, the yellow-colored "testlabel" is added as a demonstration.

1.2 Contribution

1. **Develop an improved method to calculate string similarity**

   The main contribution of this project is an approach to improve the overlap coefficient method for calculating the string similarity. Preliminary experiments have shown that the improved approach performs better than the original method.

2. **Create a foundation for an automatic labeling project**

   Labeling all issues manually is complex and people sometimes forget to label the issues, so there are lots of issues that have not been labeled yet in GitHub. To try to help reduce people's workload and label all the unlabeled issues, it would be helpful if the issues could be automatically labeled by text mining techniques. Automatically labeling the issues requires analyzing the feature of

[2]https://www.atlassian.com/git/tutorials/what-is-version-control
[3]https://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/
[4]https://guides.github.com/features/issues/
[5]https://discuss.gogs.io/t/how-to-use-predefined-label-templates/599

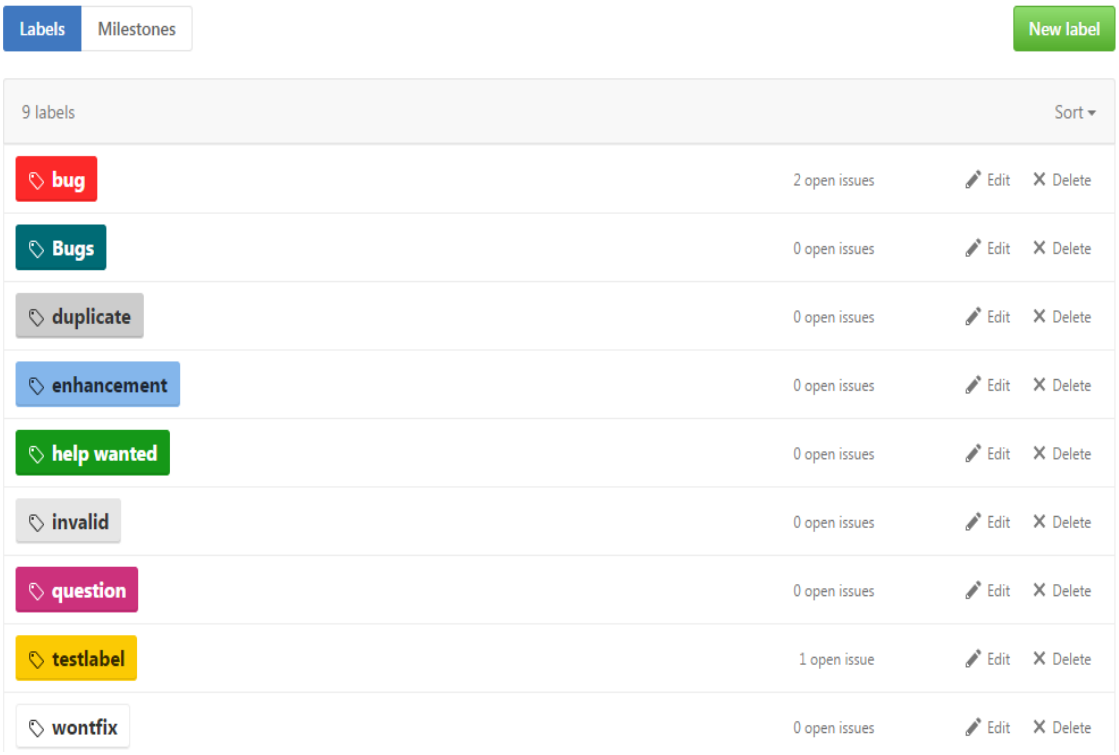

Figure 1.1: GitHub issue labels management system

the issue text and predicting the label by machine learning from existing data. This project clusters the morphological synonym labels to complete some data preprocessing work.

3. **Support management of GitHub labels and issues**

   Currently, there are many different labels with the same meaning. It is hard for people to manage the issues. In addition, it is difficult to know which kind of issue labels are more important and are used more frequently. Clustering the labels will make the label system easy to manage and allow users to know the importance of each label.

1.3 Agenda

This project report has 5 chapters in total. The following is the structure of the remaining report and a brief outline is given for each section.

Chapter 2 provides a literature review of related works from other researchers. It

includes some existing label clustering works and the approaches for string similarity calculation.

Chapter 3 introduces our approach to cluster the labels. The improved method to calculate the string similarity and the clustering approach will be described here.

Chapter 4 describes the experiment set up and the way to conduct the three experiments. Afterwards, it shows the experiment results and analyzes them based on the visualization charts.

Chapter 5 explains how the experiments answer the research questions and mentions limitation of the new method as well as the future work.

Chapter 2

Related Work

In this chapter, the concept of folksonomies are firstly introduced, and some related work will be described in the field of label clustering. In addition, a survey of the synonyms clustering approach and several commonly-used string similarity metrics will be provided.

2.1 Folksonomies

The concept of folksonomies was coined by two American information experts, T. V. Wal and G. Smith, in 2004 [18]. A folksonomy is a system in which users apply public tags to online items, typically to aid them in re-finding those items. With the development of Web 2.0, folksonomies became more and more popular. The following is a list of some popular websites that have the tag or label systems[1]:

- del.icio.us: social bookmarking web service
- Flickr: shared photos
- Instagram: online photo-sharing and social networking service
- WordPress: blogging tool and Content Management System
- Pinterest: photosharing and publishing website

[1] https://en.wikipedia.org/wiki/Folksonomy

Figure 2.1 and Figure 2.2 show the del.icio.us[2] and Flickr[3] tag systems. However, there is no limitation of the words when users tag items. As a result, these folksonomies system all have a huge amount of duplicating meaning tags, which make searching and managing the resource more difficult. Grigory Begelman et al. believe that cluster tags can improve the user experience of using the tagging system in general [2]. They collected tag data from the del.icio.use and RawSugar website. By doing the experiments, they proved that clustering techniques can and should be used with the tagging system to enhance the user experience and to improve the collaborative tagging service performance [2].

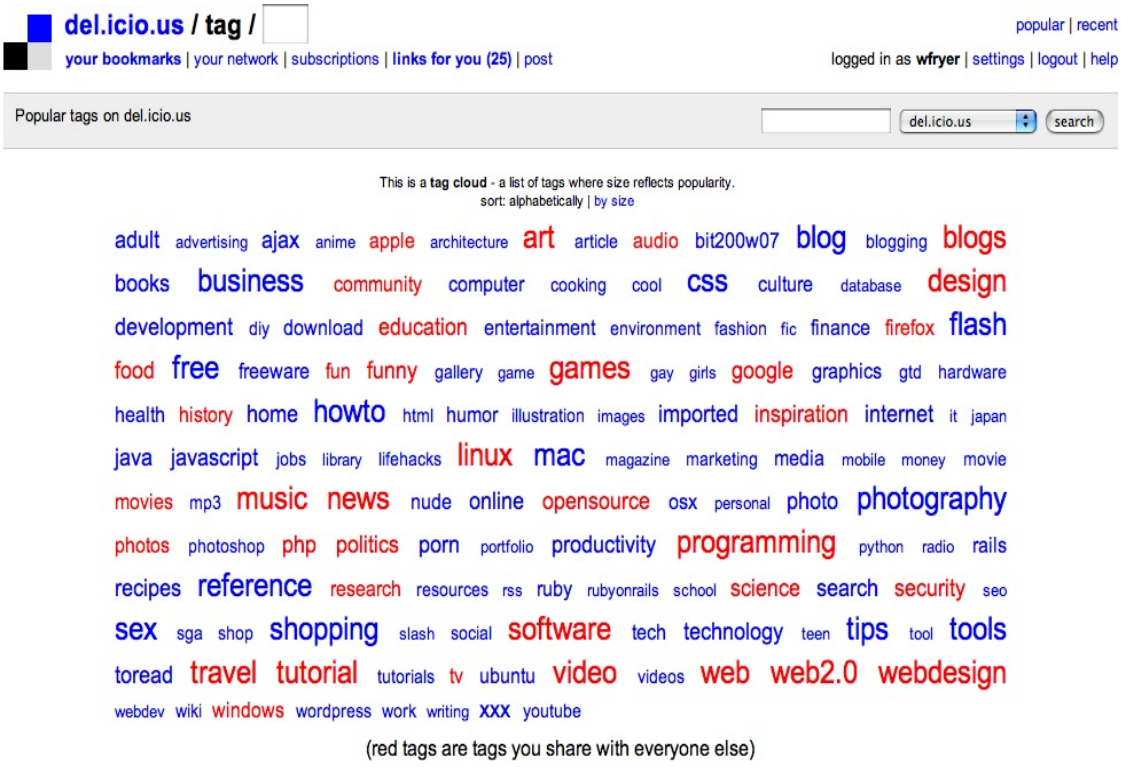

Figure 2.1: Popular tags on del.icio.us

However, few research works are related to GitHub labels. Although J. L. C. Izquierdo et al. made a visualization tool called "GitHub Label Analyzer (GiLA)," [12] this tool only aims to help people analyze and understand the GitHub label usage situation. For instance, it can use charts to visualize the most frequently used labels and the labels that are commonly used together as well as the user involvement and

[2]https://gigaom.com/wp-content/uploads/sites/1/2013/09/433566428_02cb8540a1_o.jpg
[3]http://www.formfollowsbehavior.com/wp-content/uploads/2007/03/flickr.png

Figure 2.2: Tag system on Flickr

typical label time line [12]. Figure 2.3 shows an example of the label usage situation[4] from the GiLA website.

2.2 Synonyms clustering

Many researchers have researched clustering synonyms approaches. Aron Henriksson et al. used distributional semantics to extract medical terms synonyms from the corpus of clinical text [9]. His method was to combine Random Indexing [16] and Random Permutation [24]; in 340 synonym pairs, 44% was successfully extracted [9]. Yutaka Matsuo et al. clustered words based on a word similarity measure by web count [19]. They proposed an unsupervised algorithm; a co-occurrence matrix was produced when they queried each pair of words. The newman clustering algorithm was used for identifying the word cluster. They believe that this algorithm can potentially enhance many natural language processing applications [19]. Tri Dao, et al. trained a neural network, based on a large corpus, to try to recognize the synonyms of a word from its word vector. Afterwards, they used the k-means algorithm to check whether synonyms have a statistically significant word vector similarity [4]. They found that increasing the length of word vectors negatively correlates with synonym recognition [4].

[4]http://som-research.github.io/gila/

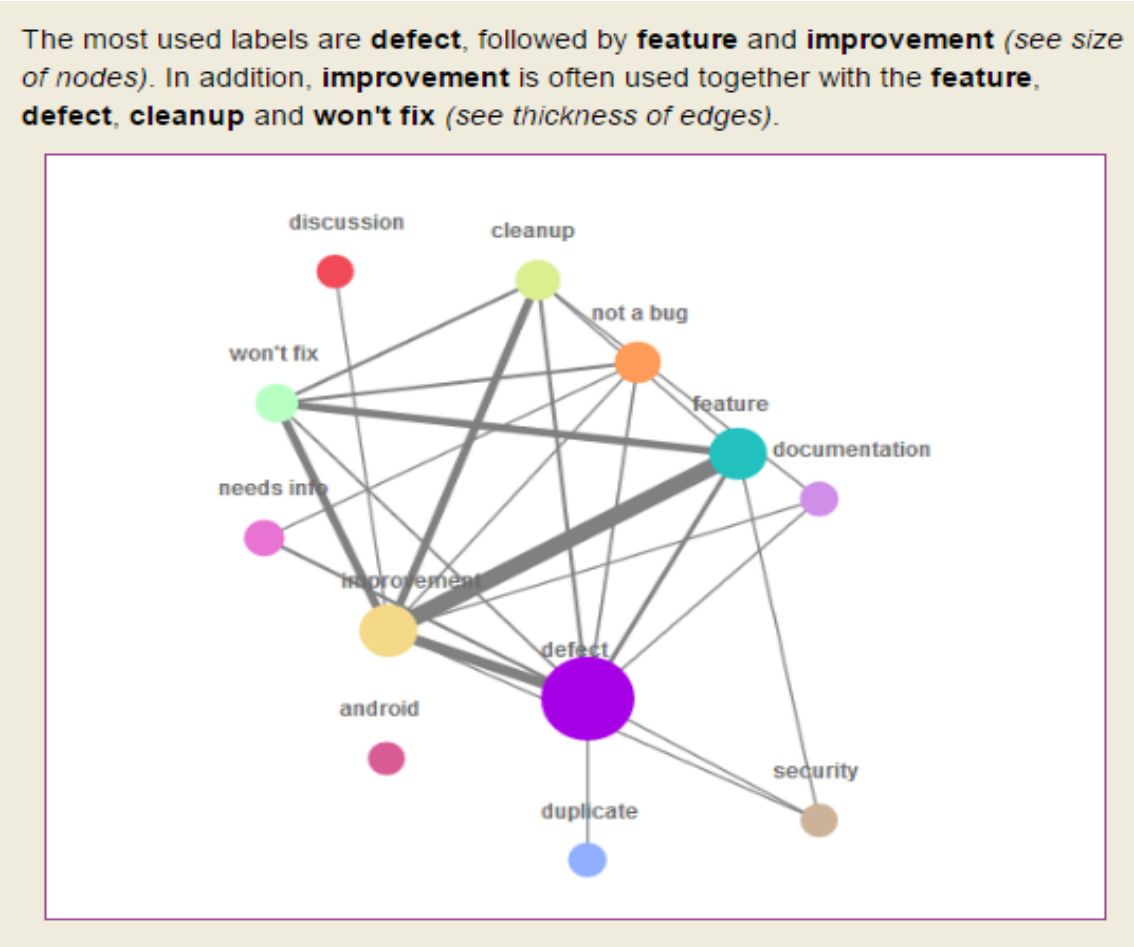

Figure 2.3: GitHub Label Analyzer: label usage situation

2.3 Similarity metrics

This project will focus on the morphological similarity of the GitHub labels. Measuring the similarity of two sets is essential for clustering. Many researchers have proposed different similarity metrics. This section will provide a summary of the commonly-used methods for calculating the string similarity when comparing two strings. By using the similarity metrics, the morphologically similar strings can be clustered. In general, the similarity metrics can be divided into two partitions; one is term-based and the other one is character-based [6].

2.3.1 Term-based similarity metrics

- Overlap coefficient

The overlap coefficient is defined as the ratio of the intersection of the two sample sets to the minimum size of the two sets [26]. Thus, if one set is a sub set of another, then the similarity coefficient will be 1. Assume the overlap coefficient of sets A and B is $O(A, B)$.

$$O(A, B) = \frac{|A \cap B|}{Min(|A|, |B|)} \tag{2.1}$$

- Jaccard similarity coefficient

  The Jaccard similarity coefficient is defined as the ratio of the intersection of sample sets to the union of the sets [13]. Assume there are two sets A and B. Their Jaccard similarity coefficient $J(A, B)$ is defined as follows:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.2}$$

- Sorensen-Dice coefficient

  The Sorensen-Dice coefficient is defined as twice the ratio of the intersection of the two sample sets to the total numbers of the two sets [25, 5]. Assume the Sorensen-Dice coefficient of two sets, A and B, is $S(A, B)$.

$$S(A, B) = \frac{2|A \cap B|}{|A| + |B|} \tag{2.3}$$

- Euclidean distance

  The Euclidean distance is another vector-based similarity metric. It sums up all the squared difference of the corresponding elements in two vectors and then calculates the square root of this sum [3]. Assume there are two vectors, p and q, and their Euclidean distance formula is $ED(p, q)$.

$$ED(p, q) = \sqrt{\sum_{k=1}^{n}(p_k - q_k)^2} \tag{2.4}$$

- Cosine similarity

  The cosine similarity is a frequently-used vector based similarity measurement approach [20]. In order to use this method, the input string should be trans-

formed to vector space. Basically, it measures the cosine of an angle created by two vectors [20]. Given two vectors, p and q, their cosine similarity $CS(p, q)$ is defined as follows:

$$CS(p, q) = \frac{p \cdot q}{|p||q|} \tag{2.5}$$

2.3.2 Character-based similarity metrics

- Levenshtein distance

  The Levenshtein distance is defined as the minimum cost of the actions (e.g., insertions, deletions, substitutions) needed to transform one string into another [8]. Dynamic programming is used as the approach to obtain this distance metric. Assume there are two strings, S_1 and S_2, the logic to calculate the Levenshtein distance $L(S_1, S_2)$ [17] is shown as follows:

  If min(i,j) = 0

$$L_{S_1,S_2}(i, j) = max(i, j) \tag{2.6}$$

  Otherwise,

$$min \begin{cases} L_{S_1,S_2}(i-1, j) + 1 \\ L_{S_1,S_2}(i, j-1) + 1 \\ L_{S_1,S_2}(i-1, j-1) + 1(S_{1i} != S_{2j}) \end{cases} \tag{2.7}$$

- Jaro distance

  The Jaro distance was proposed by Jaro in 1989 [14] and 1995 [15]. Assume there are two strings, S_1 and S_2, m is the number of matching characters and t is the number of transpositions.

  Matching characters: The character in S_1 and the character in S_2 are same and not farther than $max(|S_1|, |S_2|)/2 - 1$.

  Transpositions: Half the number of characters which are match but not in the same sequence.

  The following equations define the Jaro distance of string S_1 and S_2.

  If $m = 0$

$$J(S_1, S_2) = 0 \tag{2.8}$$

Otherwise,

$$J(S_1, S_2) = \frac{1}{3}(\frac{m}{|S_1|} + \frac{m}{|S_2|} + \frac{m-t}{t}) \tag{2.9}$$

- Jaro-Winkler distance

  The Jaro-Winkler distance is based on the Jaro distance and extends it by giving a bonus when two words share a common prefix [27]. Assume string S_1 and S_2 have common prefix length, L. Winkler also defines a prefix scale here as ρ. The maximum number of L and ρ is 4 and 0.25 respectively. Assume we have the Jaro distance $J(S1, S2)$ already, so their Jaro-Winkler distance $JW(S_1, S_2)$ is defined as follows:

$$JW(S_1, S_2) = J(S_1, S_2) + L\rho(1 - J(S_1, S_2)) \tag{2.10}$$

- Hamming distance

  The Hamming distance can only be used for comparing the strings with exactly the same length. It represents the number of places where the corresponding characters are different [7]. It describes the minimum character changes in order to transform one string into another. For instance, the hamming distance of "bag" and "bad" is 1.

2.4 Summary

This chapter shows a literature review of related works from other researchers including some existing label clustering works and several metrics of string similarity. Many research works focus on the labels of social media websites (e.g., del.icio.us and Flickr). Although the project "GitHub Label Analyzer" is close related to GitHub labels, it was designed to help people understand the label usage situation. In terms of the techniques been used, some existing works are more suitable for semantic synonyms. However, this project focuses on the morphological synonyms, which requires string similarity to be the distance measurement of clustering. Therefore, in this chapter, some commonly used string similarity metrics are also reviewed and discussed. The next chapter will describe an approach for clustering the morphological synonyms.

Chapter 3

Our Approach

This chapter describes the approach used to cluster the GitHub labels. Essentially, the labels need to be preprocessed first. After that, the English characters will be clustered by the string similarity calculation algorithm. Figure 3.1 shows the overall process. The first section will describe the six steps to preprocess a label. The next section will introduce the morphological similarity coefficient and the principle to calculate this coefficient. In addition, the method to improve the original approach is introduced. The last section presents the approach of the clustering procedure.

3.1 Label Preprocessing

Prepossessing is an important and essential step before using the data for analysis. In this case, the GitHub label name will go through a series of steps to be preprocessed. After that, it will be used for calculating the string similarity. The main purpose of this prepossessing is to get the most important part of a label, which can represent its meaning, by removing the unrelated characters. Figure 3.2 gives a brief overview of the prepossessing workflow.

Step 1: Filter English letters

As most of the label names of GitHub are in English, this project only focuses on English words. Therefore, strings like other languages, numbers or marks are not considered. In this step, all the non-English letter characters will be filtered out. For example: “0 - Backlog” will be “Backlog”. “1” will become an empty string.

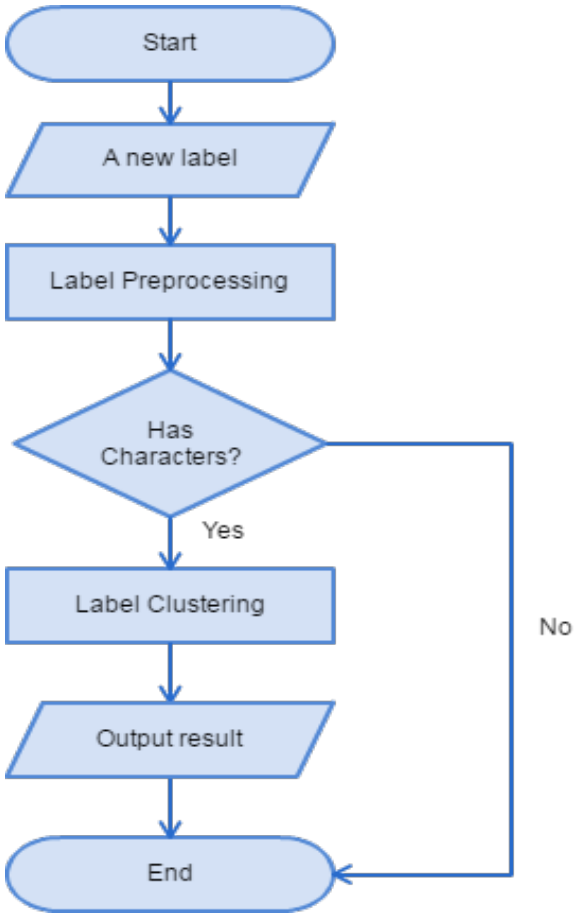

Figure 3.1: Overall process

Step 2: Delete meaningless prefixes

It can be seen that numerous label names start with a prefix. For instance: "type: feature". In this case, the prefix "type" is not the key word to be considered. The "feature" is the important word, which represents the meaning of the label. Thus, it is necessary to delete the prefix so that it will not affect the string similarity coefficient calculation result. Then, "type: feature" will become "feature". Some other examples like "status:confirmed", "info:help-needed" will be converted to "confirmed" and "help-needed".

Step 3: Delete unimportant words

By observing the label names, it can be found that some words are unimportant in terms of clustering. "Priority" is one of the most important label types in GitHub

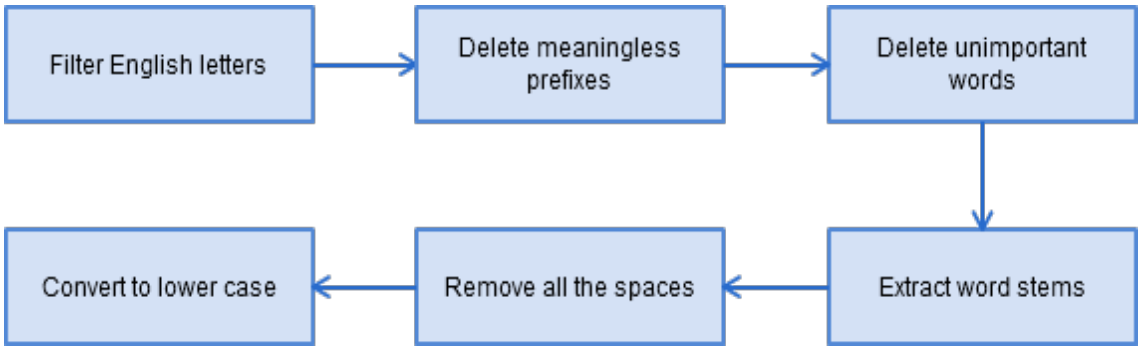

Figure 3.2: Label preprocess workflow

and it is a good example of an unimportant word. If there are two labels, "priority-high" and "priority-low", what we really care about is high and low. In other words, high and low are key words to represent the labels. Thus, we can consider the word "priority" unimportant and delete it, so that it will not affect the string similarity calculation.

Step 4: Extract word stems

The Porter stemming algorithm[1] is famous and commonly used in the natural language processing field. By using the porter stemming algorithm, the stem of a word can be acquired by removing the suffixes [21]. Normally, the words with the same stem have the similar meanings. Obtaining the stem can make the clustering algorithms perform better, because we only care about the key part of the word, which is identical and can be used to distinguish it from other words. For example, "documentation" will become "document" after being reduced to its stem.

Step 5: Remove all the spaces

Spaces also need to be removed to guarantee the accuracy of the result. Sometimes people may type two spaces between two words. Besides, depending on people's preference, some people may use "to do", while some people prefer to use "todo". Deleting the spaces can help the clustering algorithm focus on the English characters only and increase the accuracy of the clustering result.

[1]https://sourceforge.net/p/porterstemmer/code/HEAD/tree/trunk/Porter%20Stemmer/

Step 6: Convert to lower case

This step will convert all letters to lower case. Obviously, there are almost no difference between upper case and lower case. Sometimes, it is just user preference. Using only lower case letters to calculate the similarity will make the clustering result better.

3.2 Morphological Similarity Coefficient

Comparing the similarity of two strings is easy for humans, but hard for computers. Therefore, the Morphological Similarity Coefficient (MSC) is introduced to determine whether two strings are morphologically similar or not. The range of MSC is [0, 1], so the closer the MSC is to 1, the more similar these two strings will be.

This section will introduce the dynamic programming first, and then describe the method to calculate the longest common substring. At last, the limitation of overlap coefficient is analyzed and a new approach is developed to calculate the similarity coefficient.

3.2.1 Dynamic Programming

Since Dynamic Programming (DP) will be used to calculate the longest common substring in the next subsection, this subsection will give a basic introduction to it. The basic idea of DP is to break the big problem into a sequence of small problems, and each small problem uses the result of its previous problem to reduce the duplicate calculation [1]. A good example of DP is to get the N^{th} number of Fibonacci sequence [11]. A Fibonacci sequence is defined as follows:

$$F_1 = 1, F_2 = 1, F_n = F_{n-1} + F_{n-2} (n \geq 3, n \in N^*) \tag{3.1}$$

If F_5 is needed and recursion is used to solve this problem, then,

$$F_5 = F_4 + F_3 = (F_3 + F_2) + F_3 = ((F_2 + F_1) + F_2) + (F_2 + F_1)$$

As we can see that F_3 is calculated by F_2 and F_1, but it is calculated twice. The DP solution calculates each number only once. In order to calculate F_5, it keeps an array

A with 5 elements, each element is the corresponding number in Fibonacci sequence.

$$A[0] = 1, A[1] = 1,$$
$$A[2] = A[1] + A[0] = 2$$
$$A[3] = A[2] + A[1] = 3$$
$$A[4] = A[3] + A[2] = 5$$

Therefore, the answer of F_5 is 5. By using DP, each number is calculated only once. It reduces the time complexity dramatically especially when the number of N is big.

3.2.2 Longest Common Substring

The Longest Common Substring (LCS) needs to be calculated for obtaining the MSC. The LCS is the length of the longest common string which is substring of two given strings. For example,

$$(\text{"}docs\text{"}, \text{"}document\text{"}) => return\ 3$$
$$(\text{"}testing\text{"}, \text{"}test\text{"}) => return\ 4$$

Given two strings (S_1, S_2) as the parameters, Algorithm 1 describes the principle of obtaining the LCS [10]. The first step is to calculate a two-dimensional array that array[i][j] indicates the current longest common substring at the index i of S_1 and index j of S_2. There are two cases:

- $S_1.charAt(i-1)$ is equal to $S_2.charAt(j-1)$, then array[i][j] = array[i-1][j-1] + 1.
- $S_1.charAt(i-1)$ is not equal to $S_2.charAt(j-1)$, then array[i][j] = 0.

After obtaining this array, the remaining work (from line 12 to the end) is to find the maximum value inside this array.

Algorithm 1 Calculate the LCS of String S_1 and S_2

Require: $S_1 \neq null \quad \&\& \quad S_2 \neq null$

```
1: lenth1 ← S1.length
2: lenth2 ← S2.length
3: for i ∈ {1, ..., length1} do
4:     for j ∈ {1, ..., length2} do
5:         if S1.charAt(i − 1) == S2.charAt(j − 1) then
6:             array[i][j] ← array[i − 1][j − 1] + 1
7:         else
8:             array[i][j] ← 0
9:         end if
10:    end for
11: end for
12: max ← 0
13: for i ∈ {1, ..., length1} do
14:    for j ∈ {1, ..., length2} do
15:        if array[i][j] > max then
16:            max ← array[i][j]
17:        end if
18:    end for
19: end for
```

Given calculating the LCS of "docs", "document" as an example[2]. The following table shows the result of the array. Since 3 is the maximum value, the LCS of "doc" and "document" is 3.

		d	o	c	u	m	e	n	t
	0	0	0	0	0	0	0	0	0
d	0	1	0	0	0	0	0	0	0
o	0	0	2	0	0	0	0	0	0
c	0	0	0	3	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0

Table 3.1: An example of longest common substring calculation

[2]http://algorithms.tutorialhorizon.com/dynamic-programming-longest-common-substring/

3.2.3 Weighted Overlap Coefficient

Assume there are two strings, S_1 and S_2. Their lengths are L_1 and L_2 respectively. After applying the LCS algorithm, the longest length is L_{lcs}. Firstly, the overlap coefficient method will be used for calculating MSC.

As mentioned in Chapter 2, the overlap coefficient is defined as the ratio of the intersection of the two sample sets to the size of the smaller set. In this case, the L_{lcs} will be the intersection of the two strings, and the size of the smaller set will be the smaller string length. Assume the MSC calculated by the overlap coefficient is MSC_O.

$$MSC_O = \frac{|L_{lcs}|}{Min(|L_1|, |L_2|)} \tag{3.2}$$

This approach only considers the minimum value of the two sets and drops the maximum value. However, to some degree, the maximum value is also important. It is obvious that the lower maximum values are better. For instance, the LCS of two strings "test" and "highest" is 3.

$$MSC_O = \frac{|L_{lcs}|}{Min(|L_1|, |L_2|)} = \frac{|L_{lcs}|}{|L_1|)} = 0.75$$

$$\frac{|L_{lcs}|}{Max(|L_1|, |L_2|)} = \frac{|L_{lcs}|}{|L_2|)} \approx 0.43$$

Since MSC_O is high, they may be considered similar words, but they are not. Therefore, a new approach for calculating the MSC, called the weighted overlap coefficient, is proposed here. Basically, the idea is to add a weight to both the minimum and maximum values. Assume the weight is w for the bigger number, and $1 - w$ for the smaller number. Assume the similarity coefficient of set A and B is $WO(A, B)$.

$$WO(A, B) = w\frac{|A \cap B|}{Min(|A|, |B|)} + (1 - w)\frac{|A \cap B|}{Max(|A|, |B|)} \tag{3.3}$$

Assume the MSC calculated by the weighted overlap coefficient is MSC_{WO}.

$$MSC_{WO} = w\frac{|L_{lcs}|}{Min(|L_1|, |L_2|)} + (1 - w)\frac{|L_{lcs}|}{Max(|L_1|, |L_2|)} \tag{3.4}$$

The research questions are: "***Is the performance of the weighted overlap coefficient method better than the overlap coefficient?***" And "***How does the new approach compare to other similar existing similarity calcula-***

tion methods?" The following experiments and the discussion will answer these questions.

3.3 Clustering approach

After calculating the MSC of the two strings, the question is how to decide whether or not the two strings belong to one cluster. In order to make this decision, a threshold value is introduced. When the similarity coefficient is greater than the threshold, these two strings can be classified as one cluster. The range of the threshold value should be the same as the range of the MSC, which is [0, 1]. It is obvious that different threshold values will lead to different cluster quality results. The following experiments will find the different performance results by using different thresholds.

Each label will be converted into a label model including an original name and a processed name. The processed name is used for calculating the MSC. A center label is the label that represents its cluster. The following steps describe the approach to cluster label names:

1. Preprocess the label and obtain the processed label name.
2. Calculate the MSC of the label with each center label.
3. Find the maximum MSC. If there is a tie, choose the last one.
4. If this maximum MSC is greater than, or equal to, the designed threshold, the label is clustered successfully. Otherwise, it fails and it will become a new center label.
5. If there are unclassified labels, go to step 2.

As shown in Figure 3.3, the blue colored marks are center labels that represent their clusters. All the grey marks are the elements in the clusters. The red dot is the new element that needs to be clustered. The MCS of the red dot to each blue element is calculated and the maximum one is found (in the graph, it is the minimum distance). In this example, this value is greater than the threshold; thus, it is categorized to the dot cluster successfully.

Figure 3.4 shows another situation in which the maximum MSC of the red pentagon with other center labels is less than the threshold. Therefore, the red pentagon

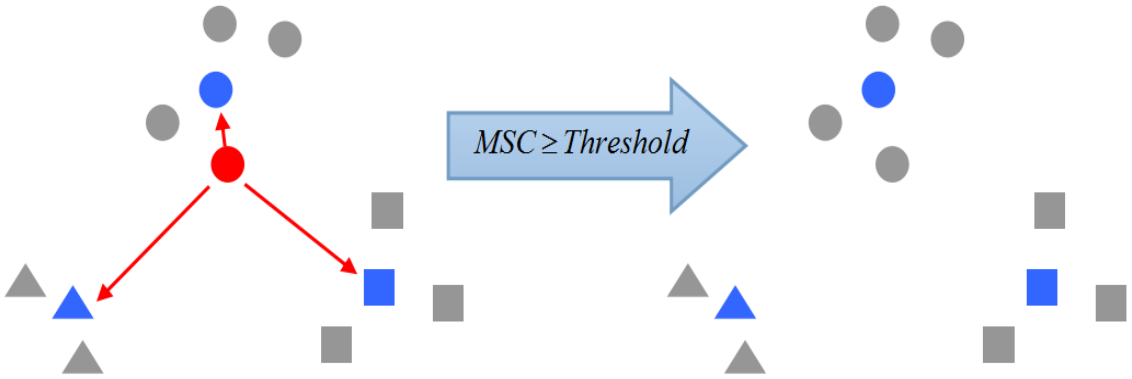

Figure 3.3: Clustering approach: MSC is greater than or equal to Threshold

does not belong to any of the existing clusters and it should be a new center label of another cluster (marked as blue).

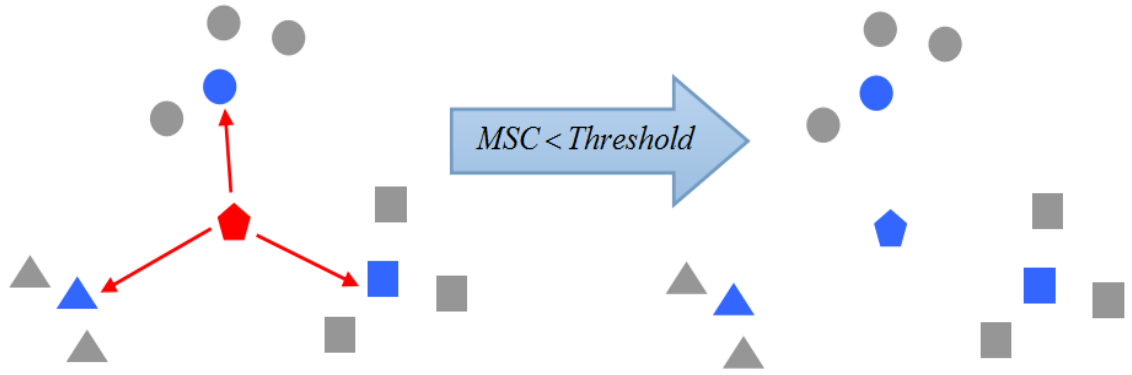

Figure 3.4: Clustering approach: MSC is less than Threshold

3.4 Summary

This chapter presents our approach for GitHub label clustering problems. The Morphological Similarity Coefficient (MSC) is significant to cluster the labels. Initially, six steps are needed in the preprocessing phase. Subsequently, a weighted overlap coefficient method is proposed to improve the overlap coefficient for calculating the MSC. Finally, a label's MSC is calculated with the center label of each cluster to find a maximum MSC. Then, the maximum MSC is compared with a threshold value to determine whether the label can be categorized successfully or not. Since a new approach (i.e., weighted overlap coefficient) is proposed, the questions are: "***Is the***

performance of the weighted overlap coefficient method better than the overlap coefficient?" And "***How does the new approach compare to other similar existing similarity calculation methods?***" The next chapter will describe the experiments and show the results to answer these questions.

Chapter 4

Experiments

In this chapter, three experiments are conducted in order to evaluate the improvement brought out by our approach. The first section will describe the experiments' set up. The evaluation approach of the clustering result as well as the method of conducting these experiments will be described in section 2 and section 3 respectively. The last section will illustrate and analyze the experiment results.

4.1 Experiments Set up

4.1.1 Data Source

This experiment uses the GHTorrent Data MySQL dump 2015-09-25[1]. The GHTorrent Data is a project that aims to create a mirror data which is scalable and queriable based on the GitHub REST API data. The GitHub contents and their dependencies are retrieved during each event by monitoring GitHub's public event time line. Afterwards, the raw JSON responses are stored to a MongoDB database and also the structure is extracted to a MySQL database[2].

4.1.2 Label Names

The label names can be obtained in the "repo_labels" table. In order to group all the label names and sort them in descending order by their frequency of usage, the following SQL command is executed:

[1]http://ghtorrent.org/downloads.html
[2]http://ghtorrent.org/

SELECT NAME AS NAME,COUNT(NAME) AS number FROM repo_labels GROUP BY NAME ORDER BY number DESC

The sorted result of the top 200 labels is listed in Table A.1. The top 200 labels have been used 906185 times and account for 90.62% of the total labels. "Enhancement" is the most frequently used label with 178434 usages and "type-feature" ranks 200^{th} with 78 usages. Considering the difficulty of doing the experiments, these top 200 label names are collected as the experiments' sample data set to be classified.

The correct clusters of these 200 labels are classified manually and the following table shows some sample label clusters:

discussion, dicsuss
Frontend, front-end
documentation, docs, doc
In progress, in-progress
medium, priority-medium, MediumPriority, Priority: Medium

Table 4.1: Sample correct label clusters

4.2 Evaluation Method

Evaluation is essential in the fields of natural language processing, data mining and machine learning. Normally, Precision, Recall and F-Measure [22] are adopted as the evaluation criteria. In order to understand Precision, Recall and F-Measure, a confusion matrix [23] needs to be introduced. The following table describes the confusion matrix:

	Predicted: Yes	**Predicted: No**
Actual: Yes	True Positive (TP)	False Negative (FN)
Actual: No	False Positive (FP)	True Negative (TN)

Table 4.2: Confusion matrix

Precision [22] is defined as:

$$\frac{TP}{TP + FP} \tag{4.1}$$

Recall [22] is defined as:

$$\frac{TP}{TP + FN} \tag{4.2}$$

F-Measure [22] is the harmonic mean of Precision and Recall, which is defined as:

$$\frac{2 * Precision}{Precision + Recall} \tag{4.3}$$

In this project, if the overall number of labels are N, there are $N * (N - 1)/2$ label pairs in total. Subsequently, the label pairs can be understood as a series of decisions. Therefore, TP TN FP FN can be defined as follows[3]:

- TP: Assign two similar labels to the same cluster.
- TN: Assign two dissimilar labels to different clusters.
- FP: Assign two dissimilar labels to the same cluster.
- FN: Assign two similar labels to different clusters.

The decision tree in Figure 4.1 can make the definitions more clear.

4.3 Experiments Description

In order to answer the research questions, three case studies are part of our evaluation.

- Case Study 1

  The purpose of this case study is to discover the performance of the overlap coefficient method. Considering that different threshold values will lead to different results, the threshold values from 0 to 1 (with 0.05 step increments in between) are picked to run the clustering algorithm by overlap coefficient. The TP, FN, FP, TN need to be recorded and the Precision, Recall and F-Measure will be calculated. Afterwards, the result will be visualized by a line chart.

[3]https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html

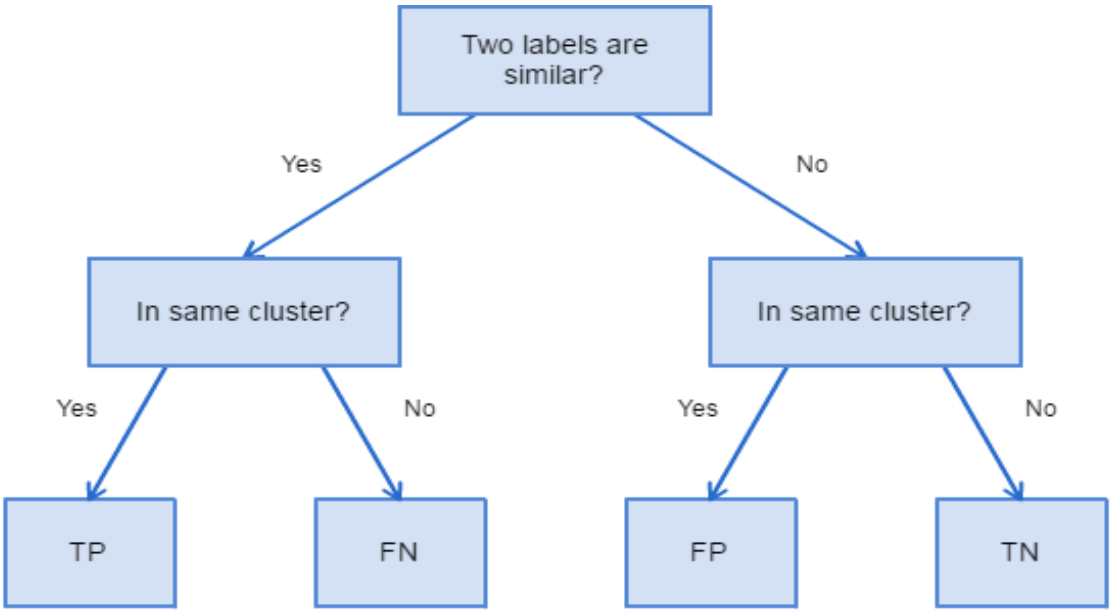

Figure 4.1: Decision tree for result evaluation

- Case Study 2

  This case study aims to compare the improved method's performance with the original method. Since both weight and threshold values can affect the F-Measure result, different weight values from 0 to 0.95 (with 0.05 step increments in between) will be picked for the experiment. For each weight value, different threshold values will be attempted, which are exactly the same as Case Study 1. By combining the weight and the threshold value together, each of them will yield a F-Measure value by applying the new weighted overlap coefficient approach. All F-Measure values will be recorded and the performance will be compared with the overlap coefficient approach.

- Case Study 3

  This case study aims to compare the performance of the weighted overlap coefficient with some other similar coefficient calculation algorithms. As shown in the literature review, there are two other similar approaches to calculate the similarity coefficient. One is the Jaccard similarity coefficient, the other one is the Sorensen-Dice coefficient.

  Jaccard similarity coefficient and Sorensen-Dice coefficient will be used as the comparison algorithms. The different threshold values (the same as the last

two experiments) will also be applied to each algorithm. The TP, FN, FP, TN will be recorded as well, and the Precision, Recall, F-Measure will be figured. Subsequently, the performance comparison with the weighted overlap coefficient will be visualized to a bar chart.

4.4 Result and Analysis

This section shows the experiment result of the three case studies and analyzes the results based on the data visualization.

Case Study 1

The propose of this experiment is to obtain the performance result of the overlap coefficient method to cluster the top 200 GitHub label names. A partial result of this experiment is shown in Table 4.3. The whole result is displayed in Table A.2. Based on the result, the Precision, Recall and F-Measure are visualized in Figure 4.2.

The graph (Figure 4.2) illustrates the values of Precision, Recall and F-Measure by applying different threshold values from 0 to 1 (with 0.05 step increments in between). According to this graph, Precision and F-Measure experience a steady growth from 0 to 0.85. After this point, they do not change any more. Comparatively, Recall does not have obvious change from threshold 0.05 to 0.65, and after 0.75 the Recall is stable at 0.9. The peak value for F-Measure is 0.8194 from threshold 0.85 to 1.

It can be seen that the higher threshold can lead to a better result. The reason is that many similarity coefficient values are 1, which proves that the data preprocessing is essential and successful. A word is reduced to its most meaningful and significant core to calculate the similarity coefficient. We can see that values do not change from

Threshold	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**F-Measure**
0	103	0	19400	0	0.0053	1.0000	0.0105
0.25	77	17320	2080	26	0.0357	0.7476	0.0681
0.5	73	19102	298	30	0.1968	0.7087	0.3080
0.75	93	19351	49	10	0.6549	0.9029	0.7592
1	93	19369	31	10	0.7500	0.9029	0.8194

Table 4.3: Partial result of the overlap coefficient performance experiment

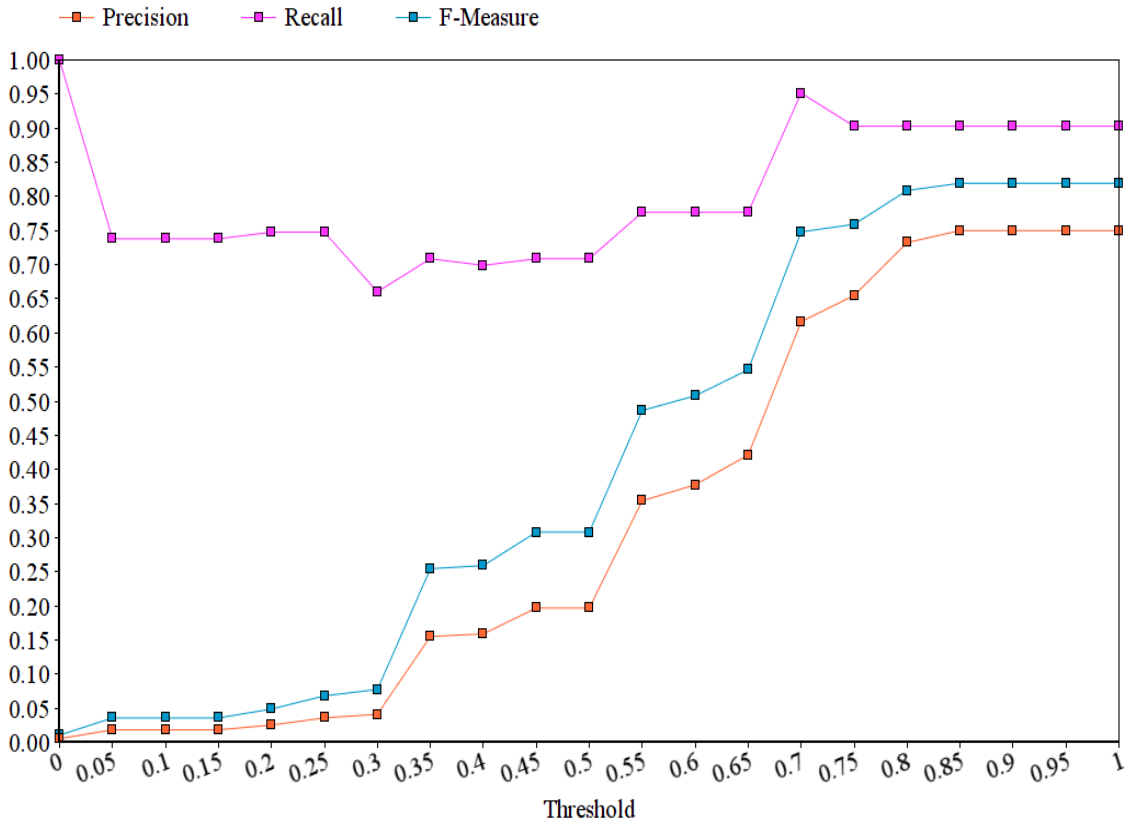

Figure 4.2: The performance of overlap coefficient method

threshold 0.85 to 1. One possible reason is that there is no similarity coefficient value in this range. This algorithm can achieve F-Measure 0.8194 as its best performance.

Case Study 2

This experiment attempts to find the best F-Measure obtainable by the new weighted overlap coefficient approach. Since both weight values and threshold values can affect this result, different weight values and threshold values are picked to run this string similarity calculation algorithm. The result is displayed in Table A.3.

Based on the result, the highest F-Measure is 0.8557, which appears in nine places. The weight and threshold values that can achieve peak performance are listed here as the w-t pair (w, t); these nine possible w-t pairs are: (0.1, 0.7) (0.15, 0.7) (0.25, 0.75) (0.3, 0.75) (0.4, 0.8) (0.45, 0.8) (0.55, 0.85) (0.7, 0.9) (0.85, 0.95). Since all threshold values are greater than 0.65, a line graph with threshold value from 0.6 to 1 is drawn in Figure 4.3.

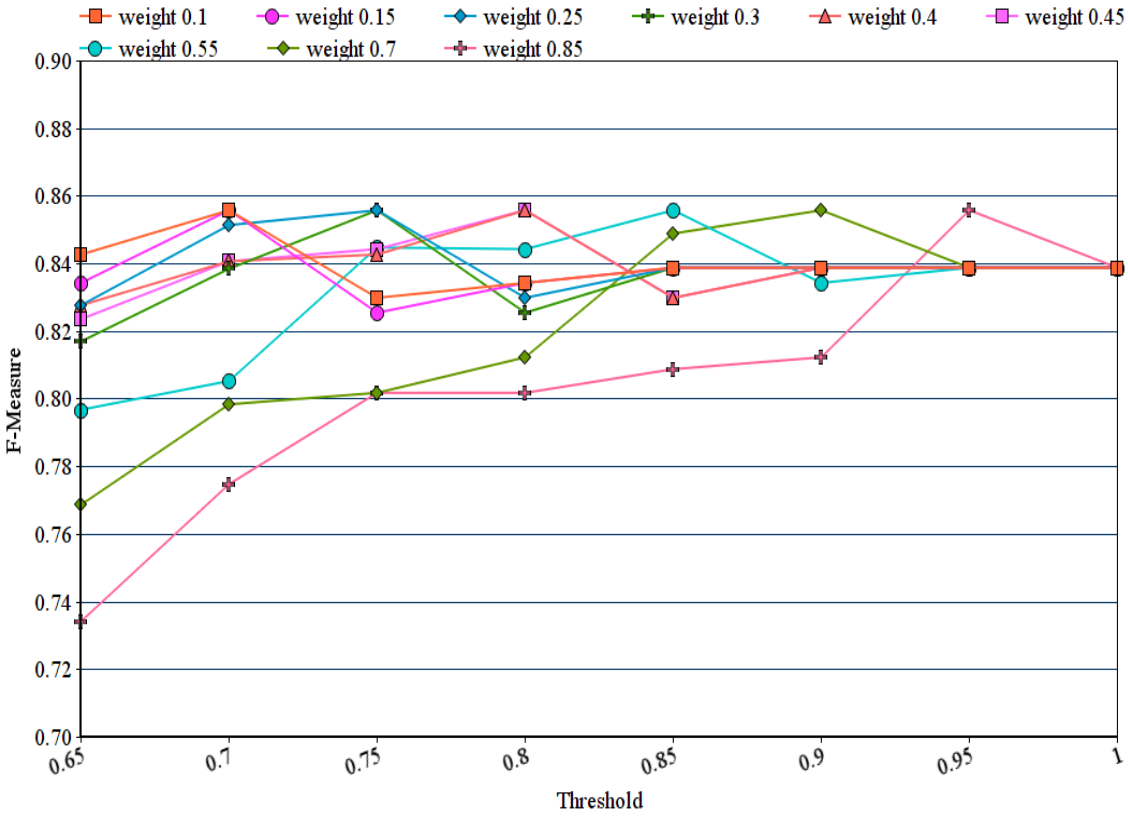

Figure 4.3: The F-Measure of weighted overlap coefficient method

As we can see from the chart, all of the nine different weight values can achieve the same peak F-Measure value (0.8557) at different thresholds. One weight value should be selected to represent the weighted overlap coefficient method for experiment purpose. Weight 0.45 and 0.55 are close to the middle (from 0.0 - 1.0). In this experiment, 0.55 was selected to represent the new approach. By comparing the experiment result and the expected clustering result, 15 labels were found to be clustered incorrectly in 198 labels (2 labels became empty strings after preprocessing), which indicates that 92.42% ((198-15)/198) of the labels were correctly clustered. In terms of the F-Measure value, the chart that compares the new approach to the overlap coefficient is shown in Figure 4.4.

Figure 4.4 illustrates the F-Measure of the weighted overlap coefficient and the overlap coefficient algorithms by applying different threshold values. By making this comparison, it is obvious that the F-Measure of the weighted overlap coefficient outperforms the overlap coefficient method regardless of the threshold values. Therefore, this experiment proves that the weighted overlap coefficient is better,

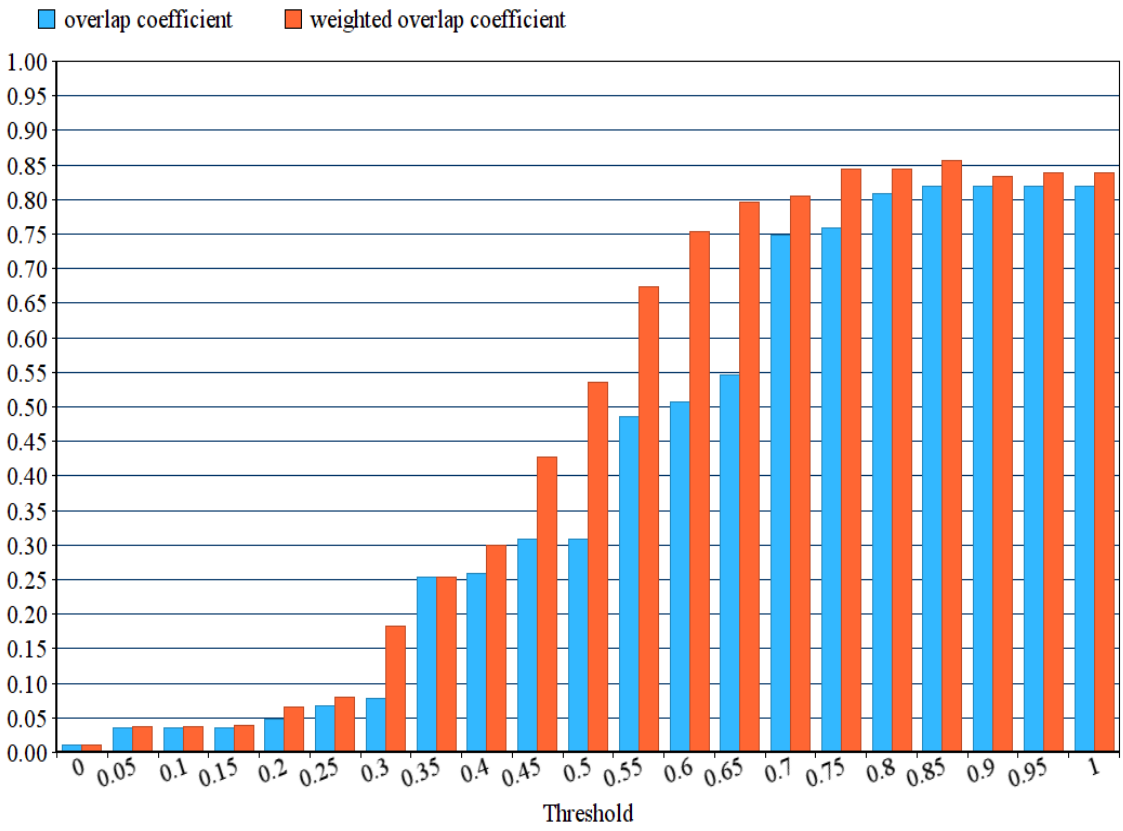

Figure 4.4: Weighted overlap coefficent and overlap coefficient comparison

and in terms of the peak F-Measure value, the performance has increased 4.43% $((0.8557 - 0.8194)/0.8194)$.

Case Study 3

The goal of this experiment is to compare the new method with the Jaccard similarity coefficient and the Sorensen-Dice coefficient. The result of clustering by the Jaccard similarity coefficient and the Sorensen–Dice coefficient are recorded in Table A.4 and Table A.5 respectively. Weight 0.55 still represents the weighted overlap coefficient approach.

Figure 4.5 compares the F-Measure values of the three similarity coefficient calculation algorithms. The chart shows that all the methods' F-Measure values experience a growth when the threshold value increases from 0 to 1. Although they reach their peak at threshold 0.85, 0.65 and 0.8 respectively, their best results are exactly the same, which is 0.8557. Additionally, it is clear from the graph that the Jaccard

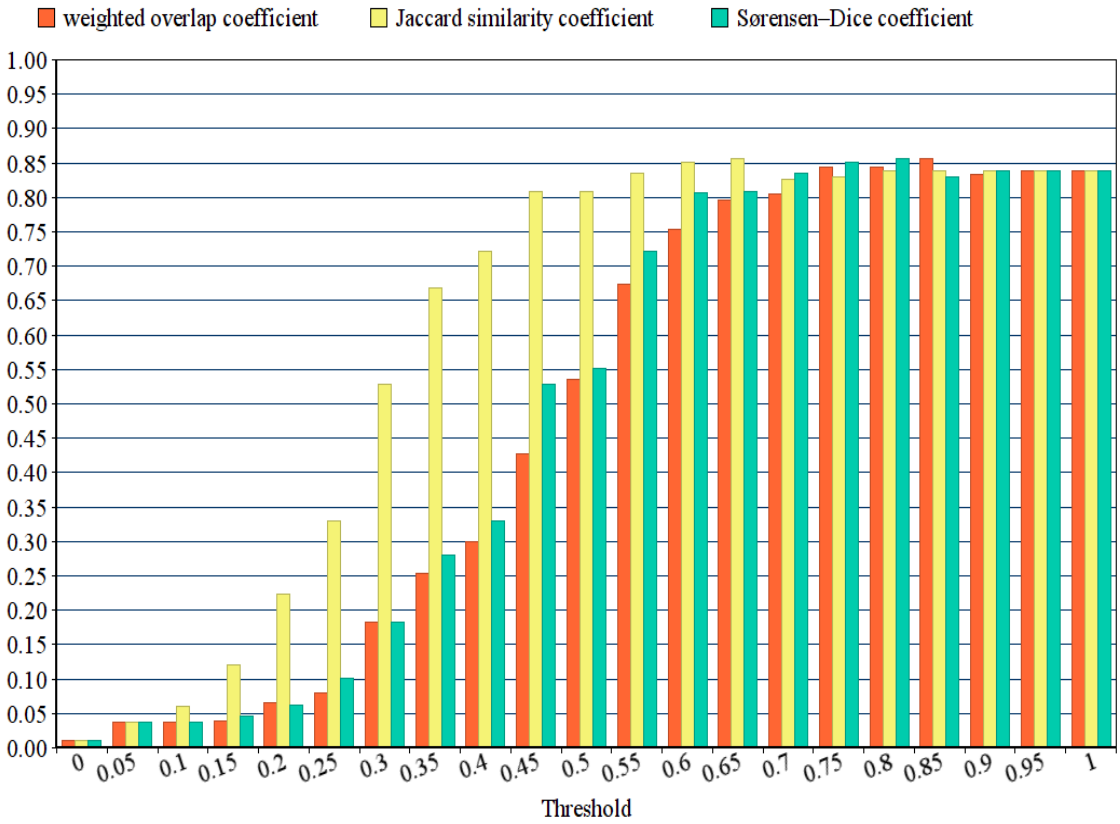

Figure 4.5: Weighted overlap coefficient, Jaccard and Sorensen-Dice comparison

similarity coefficient is the best approach due to the higher performance under each threshold value. However, there is no significant performance difference between the Sorensen-Dice coefficient and the weighted overlap coefficient.

In conclusion, the improved algorithm is able to perform the same as the Sorensen-Dice coefficient algorithm, although it is still not as good as the Jaccard similarity coefficient method.

4.5 Summary

This chapter answers the questions raised in Chapter 3 by conducting three experiments. Firstly, the experiments' set up and evaluation method are described. Afterwards, the three experiments' approaches and results are shown. The first experiment illustrates the performance of the overlap coefficient method, which can achieve 0.8194 as the best F-Measure value. The second one proves that 92.42% of the labels are correctly clustered and the new approach outperforms the overlap coefficient because

of the 4.43% increasement of the F-Measure value. The last one indicates that the new approach can perform as well as the Sorensen-Dice coefficient, but its overall performance is lower than the Jaccard similarity coefficient.

Chapter 5

Discussion

This chapter will describe how the experiments answer the research questions. Limitations and future works will also be mentioned.

This project improves the overlap coefficient algorithm by adding a weight. In order to answer the research questions, the top 200 frequently used GitHub labels are collected as the experiment data source. The correct clusters are classified manually and different methods are used to calculate the string similarity. To evaluate the classification result, the F-Measure, which is the harmonic mean of Precision and Recall, is used as the evaluation benchmark.

Figure 4.4 shows that the improved algorithm outperforms the original algorithm by increasing the F-Measure value by 4.43%. Meanwhile, 92.42% of the experiment GitHub labels, which belong to morphological synonyms, are successfully clustered by the new approach.

The new approach is compared with other similar methods, like the Jaccard similarity coefficient and the Sorensen-Dice coefficient which are mentioned in Chapter 2. Figure 4.5 reveals that the performance of the new approach is similar to the Sorensen-Dice coefficient, but lower than the Jaccard similarity coefficient in terms of the overall performance. However, regarding the peak F-Measure, all three methods can achieve the same value (0.8557).

5.1 Limitation and Future work

The previous results show that this algorithm can correctly classify a large proportion of the GitHub labels successfully. Although this algorithm can achieve a high

F-Measure, there are two major shortcomings of this string similarity calculation algorithm, which cause some labels to be incorrectly clustered.

1. The weighted overlap coefficient does not work well for the short labels. For instance, sometimes "ux" could be misclassified as the same group with "linux". The cause of this issue is that short labels can easily obtain high MSC values and be classified to wrong clusters.

2. It cannot distinguish antonyms (e.g., confirmed, unconfirmed). This algorithm will assume that they belong to a same cluster. However, they have completely different meanings. The reason is that this algorithm only considers the morphological similarity. Obviously, antonyms with a prefix (e.g., dis-, in-, un-) can still achieve high MSC results.

However, the overlap coefficient, Jaccard similarity coefficient and Sorensen-Dice coefficient should all have the same issues described as above. The reason is because all of them do not have special mechanisms for short labels and antonyms (adding a prefix).

In terms of future work, we could investigate more precise methods to calculate the string similarity metric for clustering, which can have some mechanisms to overcome the drawbacks. One limitation for this project is the small experiment data set. Conducting the case studies with a bigger data set (i.e., more GitHub labels) may lead to more accurate result.

Appendix A

Additional Information

A.1 Top 200 sorted GitHub labels

A.2 Experiment results of the three case studies

Label Name	Count	Label Name	Count	Label Name	Count
enhancement	178434	low priority	606	support	254
bug	167991	Important	575	Content	250
question	111396	minor	536	website	249
wontfix	100787	Security	525	confirmed	248
invalid	98018	core	491	gui	248
duplicate	97158	api	456	Bugs	245
Help Wanted	86127	Defect	446	story	240
Feature	8789	Urgent	397	JavaScript	240
documentation	2724	High	396	Won't Fix	233
task	2621	tests	387	Accepted	229
Feature Request	2057	major	372	priority-high	229
todo	1957	windows	372	Done	227
0 - Backlog	1140	Low	370	priority-low	225
In Progress	1138	priority	370	P1	223
new feature	1122	Backend	364	development	219
improvement	1031	wishlist	357	optimization	217
1 - Ready	957	doc	352	blocked	213
docs	940	Blocker	350	infrastructure	211
design	898	Suggestion	348	Medium Priority	207
critical	859	request	344	proposal	207
high priority	806	easy	330	future	204
3 - Done	766	cleanup	326	Requirement	204
discussion	760	research	311	Admin	201
fixed	744	medium	310	backlog	200
2 - Working	742	usability	302	P2	195
testing	683	frontend	295	Linux	190
UI	683	Android	294	ux	189
refactor	658	imported	288	unconfirmed	189
test	631	server	286	client	189
Refactoring	629	css	274	features	187
performance	628	Build	272	Database	187
ready	623	priority-medium	271	chore	187
idea	609	feature-request	255	Comments	184
code	181	resolved	181	maintenance	179

continue on next page

Label Name	Count	Label Name	Count	Label Name	Count
Plugin	179	CHANGE	123	python	93
regression	174	JS	123	Can't reproduce	93
upstream	172	type:bug	123	architecture	93
new	167	Interface	122	epic	92
meta	167	trivial	122	Patch	92
pending	167	low-priority	122	Info	91
Data	164	Not a bug	121	plugins	91
waffle:ready	163	graphics	121	RFC	89
review	162	translation	118	layout	87
Nice to have	162	HTML	117	Configuration	86
Priority: High	158	high-priority	117	1	85
Fix	153	wiki	116	Maybe	85
release	150	Issue	115	Next	84
report	149	Error	113	quality	84
P3	149	i18n	111	worksforme	83
Feedback	149	packaging	111	other	83
mobile	147	Parser	111	watson	83
iOS	144	in-progress	109	confirmed bug	83
On hold	141	WIP	107	cli	82
priority: low	141	priority:high	105	OSX	82
web	139	Easy Pick	103	Type: Bug	82
Rejected	139	investigate	103	front-end	82
blog	137	waiting	103	functionality	82
planned	132	User Interface	102	prio:high	82
Ideas	132	Type-Task	102	Search	81
Crash	131	later	101	To-Do	81
User Story	130	deployment	100	Hard	81
low priority	130	php	100	priority:low	80
To do	129	normal	96	Priority: Medium	79
compatibility	128	Priority-Critical	96	Tools	79
dev	128	Internal	96	Type-Other	79
discuss	127	style	95	type-feature	78
Update	125	Minor Bug	94		

Table A.1: Top 200 sorted GitHub labels

Threshold	TP	TN	FP	FN	Precision	Recall	F-Measure
0	103	0	19400	0	0.0053	1	0.0105
0.05	76	15320	4080	27	0.0183	0.7379	0.0357
0.1	76	15320	4080	27	0.0183	0.7379	0.0357
0.15	76	15320	4080	27	0.0183	0.7379	0.0357
0.2	77	16419	2981	26	0.0252	0.7476	0.0487
0.25	77	17320	2080	26	0.0357	0.7476	0.0681
0.3	68	17813	1587	35	0.0411	0.6602	0.0774
0.35	73	19002	398	30	0.155	0.7087	0.2544
0.4	72	19019	381	31	0.1589	0.699	0.259
0.45	73	19102	298	30	0.1968	0.7087	0.308
0.5	73	19102	298	30	0.1968	0.7087	0.308
0.55	80	19254	146	23	0.354	0.7767	0.4863
0.6	80	19268	132	23	0.3774	0.7767	0.5079
0.65	80	19290	110	23	0.4211	0.7767	0.5461
0.7	98	19339	61	5	0.6164	0.9515	0.7481
0.75	93	19351	49	10	0.6549	0.9029	0.7592
0.8	93	19366	34	10	0.7323	0.9029	0.8087
0.85	93	19369	31	10	0.75	0.9029	0.8194
0.9	93	19369	31	10	0.75	0.9029	0.8194
0.95	93	19369	31	10	0.75	0.9029	0.8194
1	93	19369	31	10	0.75	0.9029	0.8194

Table A.2: Case Study 1 - Result of the overlap coefficient performance experiment

T\W	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45
0	0.0105	0.0105	0.0105	0.0105	0.0105	0.0105	0.0105	0.0105	0.0105	0.0105
0.05	0.0342	0.0374	0.0374	0.0376	0.0376	0.0376	0.0376	0.0377	0.0377	0.0379
0.1	0.0443	0.0471	0.044	0.0376	0.0376	0.0376	0.0376	0.0377	0.0377	0.0379
0.15	0.0742	0.0611	0.0528	0.0504	0.0469	0.0461	0.0395	0.0396	0.0396	0.0396
0.2	0.0946	0.091	0.0913	0.1059	0.0802	0.0802	0.06	0.0598	0.0607	0.0607
0.25	0.1012	0.1079	0.097	0.1079	0.107	0.106	0.1062	0.1062	0.103	0.1005
0.3	0.2876	0.2484	0.2401	0.2341	0.227	0.2237	0.1826	0.1824	0.1818	0.1818
0.35	0.4686	0.4181	0.3927	0.388	0.3252	0.2857	0.2878	0.2878	0.2634	0.2535
0.4	0.5053	0.513	0.4901	0.4771	0.447	0.447	0.4147	0.4176	0.3437	0.3458
0.45	0.5733	0.5875	0.5732	0.574	0.5517	0.5517	0.5648	0.5052	0.5052	0.4562
0.5	0.5733	0.5733	0.5695	0.5875	0.6013	0.5775	0.6044	0.5757	0.5706	0.5714
0.55	0.7742	0.7742	0.7873	0.7803	0.7805	0.7538	0.7286	0.7286	0.7153	0.7153
0.6	0.7981	0.7867	0.7925	0.7925	0.8056	0.8056	0.8101	0.7967	0.784	0.784
0.65	0.8469	0.8254	0.8426	0.8342	0.8218	0.8276	0.8169	0.8093	0.8276	0.8235
0.7	0.8254	0.8298	0.8557	0.8557	0.8513	0.8513	0.8384	0.8442	0.8406	0.8406
0.75	0.8298	0.8387	0.8298	0.8254	0.8254	0.8557	0.8557	0.8513	0.8426	0.8442
0.8	0.8387	0.8387	0.8342	0.8342	0.8298	0.8298	0.8254	0.8254	0.8557	0.8557
0.85	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8342	0.8298	0.8298
0.9	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387
0.95	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387
1	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387

continue on next page

T\W	**0.5**	**0.55**	**0.6**	**0.65**	**0.7**	**0.75**	**0.8**	**0.85**	**0.9**	**0.95**
0	0.0105	0.0105	0.0105	0.0105	0.0105	0.0105	0.0105	0.0105	0.0105	0.0105
0.05	0.0379	0.0379	0.0379	0.0379	0.0379	0.0379	0.0379	0.0379	0.0379	0.0379
0.1	0.0379	0.0379	0.0379	0.0379	0.0379	0.0379	0.0379	0.0379	0.0379	0.0379
0.15	0.0382	0.0383	0.0379	0.0379	0.0378	0.0378	0.0378	0.0378	0.0378	0.0378
0.2	0.0507	0.065	0.066	0.0669	0.0655	0.0648	0.0635	0.0635	0.0635	0.0635
0.25	0.0798	0.0798	0.0667	0.0658	0.0719	0.071	0.0695	0.0695	0.0695	0.0695
0.3	0.1818	0.1818	0.1226	0.1226	0.1226	0.1177	0.1015	0.0904	0.0831	0.0831
0.35	0.2535	0.2535	0.2516	0.2516	0.2485	0.2485	0.2478	0.2478	0.2478	0.2478
0.4	0.3228	0.2994	0.2898	0.2817	0.2817	0.2817	0.2817	0.2817	0.2817	0.2817
0.45	0.4276	0.4276	0.4176	0.4167	0.3517	0.3381	0.328	0.3184	0.3184	0.3184
0.5	0.537	0.5355	0.4587	0.4587	0.4587	0.4562	0.4562	0.4562	0.4562	0.4562
0.55	0.7153	0.6735	0.6735	0.6164	0.5337	0.535	0.531	0.521	0.5179	0.5179
0.6	0.7717	0.7538	0.7101	0.7101	0.7101	0.7076	0.6469	0.5819	0.5819	0.5819
0.65	0.8033	0.7967	0.7935	0.7935	0.7686	0.7452	0.7452	0.7341	0.7127	0.6512
0.7	0.8161	0.8052	0.8017	0.8017	0.7983	0.7983	0.7949	0.7747	0.7568	0.7481
0.75	0.8488	0.8447	0.8267	0.8052	0.8017	0.8017	0.8017	0.8017	0.8017	0.8017
0.8	0.8513	0.8442	0.8488	0.8447	0.8122	0.8087	0.8087	0.8017	0.8017	0.8017
0.85	0.8254	0.8557	0.8513	0.8384	0.8488	0.8235	0.8087	0.8087	0.8087	0.8087
0.9	0.8387	0.8342	0.8298	0.8254	0.8557	0.8513	0.8488	0.8122	0.8087	0.8087
0.95	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8298	0.8557	0.8488	0.8087
1	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387	0.8387

Table A.3: Case Study 2: Selection of the best weight and threshold pair

Threshold	**TP**	**TN**	**FP**	**FN**	**Precision**	**Recall**	**F-Measure**
0	103	0	19400	0	0.0053	1	0.0105
0.05	75	15570	3830	28	0.0192	0.7282	0.0374
0.1	74	17144	2256	29	0.0318	0.7184	0.0608
0.15	80	18248	1152	23	0.0649	0.7767	0.1199
0.2	85	18827	573	18	0.1292	0.8252	0.2234
0.25	80	19097	303	23	0.2089	0.7767	0.3292
0.3	95	19238	162	8	0.3696	0.9223	0.5278
0.35	95	19314	86	8	0.5249	0.9223	0.669
0.4	95	19335	65	8	0.5938	0.9223	0.7224
0.45	87	19375	25	16	0.7768	0.8447	0.8093
0.5	87	19375	25	16	0.7768	0.8447	0.8093
0.55	84	19386	14	19	0.8571	0.8155	0.8358
0.6	83	19391	9	20	0.9022	0.8058	0.8513
0.65	83	19392	8	20	0.9121	0.8058	0.8557
0.7	78	19392	8	25	0.907	0.7573	0.8254
0.75	78	19393	7	25	0.9176	0.7573	0.8298
0.8	78	19395	5	25	0.9398	0.7573	0.8387
0.85	78	19395	5	25	0.9398	0.7573	0.8387
0.9	78	19395	5	25	0.9398	0.7573	0.8387
0.95	78	19395	5	25	0.9398	0.7573	0.8387
1	78	19395	5	25	0.9398	0.7573	0.8387

Table A.4: Case Study 3 - Result of the Jaccard coefficient performance experiment

Threshold	TP	TN	FP	FN	Precision	Recall	F-Measure
0	103	0	19400	0	0.0053	1	0.0105
0.05	75	15570	3830	28	0.0192	0.7282	0.0374
0.1	75	15570	3830	28	0.0192	0.7282	0.0374
0.15	79	16130	3270	24	0.0236	0.767	0.0458
0.2	74	17218	2182	29	0.0328	0.7184	0.0627
0.25	80	17987	1413	23	0.0536	0.7767	0.1003
0.3	84	18665	735	19	0.1026	0.8155	0.1822
0.35	76	19036	364	27	0.1727	0.7379	0.2799
0.4	80	19097	303	23	0.2089	0.7767	0.3292
0.45	95	19238	162	8	0.3696	0.9223	0.5278
0.5	96	19251	149	7	0.3918	0.932	0.5517
0.55	95	19335	65	8	0.5938	0.9223	0.7224
0.6	96	19361	39	7	0.7111	0.932	0.8067
0.65	87	19375	25	16	0.7768	0.8447	0.8093
0.7	84	19386	14	19	0.8571	0.8155	0.8358
0.75	83	19391	9	20	0.9022	0.8058	0.8513
0.8	83	19392	8	20	0.9121	0.8058	0.8557
0.85	78	19393	7	25	0.9176	0.7573	0.8298
0.9	78	19395	5	25	0.9398	0.7573	0.8387
0.95	78	19395	5	25	0.9398	0.7573	0.8387
1	78	19395	5	25	0.9398	0.7573	0.8387

Table A.5: Case Study 3 - Result of the Sorensen-Dice coefficient performance experiment

Bibliography

[1] Amir A Amini, Terry E Weymouth, and Ramesh C Jain. Using dynamic programming for solving variational problems in vision. *IEEE Transactions on pattern analysis and machine intelligence*, 12(9):855–867, 1990.

[2] Grigory Begelman, Philipp Keller, Frank Smadja, et al. Automated tag clustering: Improving search and exploration in the tag space. In *Collaborative Web Tagging Workshop at WWW2006, Edinburgh, Scotland*, pages 15–33, 2006.

[3] Per-Erik Danielsson. Euclidean distance mapping. *Computer Graphics and image processing*, 14(3):227–248, 1980.

[4] Tri Dao, Sam Keller, and Alborz Bejnood. Alternate equivalent substitutes: Recognition of synonyms using word vectors. 2013.

[5] Lee R Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[6] Wael H Gomaa and Aly A Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13), 2013.

[7] Matthew X He, Sergei V Petoukhov, and Paolo E Ricci. Genetic code, hamming distance and stochastic matrices. *Bulletin of mathematical biology*, 66(5):1405–1421, 2004.

[8] Wilbert Jan Heeringa. *Measuring dialect pronunciation differences using Levenshtein distance*. PhD thesis, Citeseer, 2004.

[9] Aron Henriksson, Hans Moen, Maria Skeppstedt, Ann-Marie Eklund, Vidas Daudaravicius, and Martin Hassel. Synonym extraction of medical terms from clinical text using combinations of word space models. In *Proc. of the 5th International Symposium on Semantic Mining in Biomedicine (SMBM 2012)*, 2012.

[10] Daniel S Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM (JACM)*, 24(4):664–675, 1977.

[11] AF Horadam. A generalized fibonacci sequence. *The American Mathematical Monthly*, 68(5):455–459, 1961.

[12] Javier Luis Cánovas Izquierdo, Valerio Cosentino, Belén Rolandi, Alexandre Bergel, and Jordi Cabot. Gila: Github label analyzer. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 479–483. IEEE, 2015.

[13] Paul Jaccard. *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Impr. Corbaz, 1901.

[14] Matthew A Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[15] Matthew A Jaro. Probabilistic linkage of large public health data files. *Statistics in medicine*, 14(5-7):491–498, 1995.

[16] Pentii Kanerva, Jan Kristoferson, and Anders Holst. Random indexing of text samples for latent semantic analysis. In *Proceedings of the Cognitive Science Society*, volume 1, 2000.

[17] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[18] Adam Mathes. Folksonomies: Cooperative classification and communication through shared metadata, 2004. *URL http://www. adammathes. com/academic/computer-mediated-communication/folksonomies. html*, 2010.

[19] Yutaka Matsuo, Takeshi Sakaki, Kôki Uchiyama, and Mitsuru Ishizuka. Graph-based word clustering using a web search engine. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 542–550. Association for Computational Linguistics, 2006.

[20] Hieu V Nguyen and Li Bai. Cosine similarity metric learning for face verification. In *Asian Conference on Computer Vision*, pages 709–720. Springer, 2010.

[21] Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[22] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[23] Foster Provost and Ron Kohavi. Guest editors' introduction: On applied research in machine learning. *Machine learning*, 30(2):127–132, 1998.

[24] Magnus Sahlgren, Anders Holst, and Pentti Kanerva. Permutations as a means to encode order in word space. 2008.

[25] Thorvald Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on danish commons. *Biol. Skr.*, 5:1–34, 1948.

[26] Dezydery Szymkiewicz. *Une contribution statistique a la géographie floristique*. Polskie Towarzystwo Botaniczne, 1934.

[27] William E Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. 1990.