Detecting Duplicate Stack Overflow Questions Exploiting the Textual Information, and a Semantic-based Tag Hierarchy

Cristian - Alexandru Botocan
Supervisor(s): Dr. Maliheh Izadi, Prof. Dr. Arie van Deursen
EEMCS, Delft University of Technology, The Netherlands

June 19, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

The users of the most widespread Software Engineering dedicated forum, Stack Overflow (SO), are confronted by the issue of posting duplicate questions and spending time waiting for an answer. Currently, only the SO users with a high reputation and the moderators manually determine this type of post. Hence, an automatic solution can save substantial time and work. As a solution, we propose a system split into three components.

First, the textual information component is an ML-based solution to decide whether a question pair is a duplicate or not by analyzing its encoded version. Additionally, we use the Doc2Vec model for question embedding, which considers the title and body as input. As a second feature, we build a tag analyzer. Lastly, we introduce a novel element for improving the results - a semantic-based tag hierarchy. To give a better overview of the usefulness of using this kind of hierarchy, we explore different hierarchies - built fully automated or manually adjusted, iterating through their construction and the number of depth levels. As baselines, we compare the results against the Gaussian Naive Bayes, Decision Tree, and K-Nearest Neighbours classifiers, analyzing only the question's pair textual information. As a result, the Logistic Regression and SVM classifiers, along with the tags and hierarchy, obtain better results than all the baselines. Our best configuration achieves a 92.10% accuracy, 91.68% recall, and 92.10% F1-score.

1 Introduction

Stack Overflow (SO) is a prominent Community-based Question Answer (CQA) website where users can ask questions regarding various Software Engineering topics. Furthermore, some posts refer to the same question answered previously. These duplicate questions lead users to spend time waiting for answers that are already accessible. At this moment, this duplicate question detection is done manually by moderators and SO users with a high reputation, and an automatic solution can save a significant amount of time and work. Moreover, another motivation is that the number of duplicate questions in the SO posts dataset might decrease.

Improving the detection of duplicate SO posts is an important topic in the Software Engineering research field, with many studies presenting different types of solutions. For instance, researchers in studies [1, 2] tried to compute a final probability using more components. Furthermore, due to the huge dimension of the SO questions dataset and the limited computation power, the studies [1, 3, 4, 5, 6, 7] divided the initial set of questions into classes, using the criteria of their most common tags, i.e., Java, C++, Python, HTML, etc. In our work, we do not apply this type of a split operation because we want to calculate the performance and utility of combining the hierarchy and tag scores. In addition, regarding the textual information which is considered for creating the question embedding, there are various options: studies [8, 9, 10, 11] considered only the post's body, while work [12] took into consideration also the question's title and others [13, 14] analyzed even the code snippets. Finally, an interesting way to improve the Github tag recommendations is to explore the semantic meaning between tags using a graph representation, a process shown in the papers [15, 16]. Since we are interested in finding duplicate questions, we apply a similar approach to SO posts.

To tackle this problem, we propose a duplicate question detector that receives a pair of SO posts and labels it as a duplicate pair or non-duplicate pair. The final classification probability is formed from the other three components. The first one represents the textual information analyzer, where an ML-based model acts as a classifier by analyzing the question pair embeddings. To build that encoded version of the pairs, we use the Doc2Vec technique, applied to both question's title and body. A heuristic tag matching scoring represents the second component.

The novelty is introduced in the third feature since we want to also analyze the tags from the semantical perspective. Therefore, we introduce a hierarchy component built based on the SED-KGraph [15]. The process of achieving the proposed feature starts by creating the hierarchy with automated techniques of generating initial clusters and then intervening manually, depending on the case.

Regarding results, all models that use the hierarchy and tags components besides the textual information obtain better results. As an overview, our best configuration achieves a 61.72%, 68.71%, and 49.27% improvements in accuracy, recall, and F1-score respectively, against the best baseline model outcomes. Moreover, using the Support Vector Machine classifier for scoring the textual information of the question embeddings pair, we obtain the best results for this component (accuracy 83.0%, recall 77.59%, F1-score 83.16%, and precision 92.8%) above the ones from the best baseline model (accuracy 56.95%, recall 54.26%, F1-score 61.7%, and precision 88.5%).

Our contribution is that we run experiments for the duplicate question detection problem using a semantic-based tag hierarchy, and the results are promising. Moreover, to better understand the performance, we create different hierarchies, using multiple automatic algorithms for clustering a knowledge graph. We conclude with the remark that the depth level of a hierarchy affects the component scoring, and a suitable depth value is between 5-10. Still, we consider that using the semantical links between the tags, in addition to the other components, can benefit. Lastly, we make our implementations available on Github[1], such that other studies can use our proposed models.

2 Problem Definition

Detecting the repetitive SO posts is a manual process done by the maintainers and SO members with high reputation. However, an automatic process will help the community do this more efficiently and avoid posting duplicate questions as much as possible. For that, we build an ML-based model that, given a pair of SO posts, can categorize it as a duplicate or non-duplicate tuple. Besides using the textual information score obtained from the questions' title and body, the duplicate detection system also considers the two other scoring components: tags and the tags hierarchy built from SED-KGraph presented in the paper by Izadi et al. [15].

The above problem, can also be formalized in a mathematical way as follows: given a pair of a SO questions q_i, q_j where $q_i = (title_i, body_i, tags_i = \{t_1, t_2, ..., t_l\})$ which contains a title ($title_i$), a textual description body ($body_i$), a list of tags $tags_i = \{t_1, t_2, ..., t_l\}$, the duplicate detection system outputs 0 if the pair is not a duplicate one, 1 otherwise. Furthermore, for tracking the usefulness of tags and semantic-based tag hierarchy, additonally to the textual information score, the system use tags score and hierarchy score.

[1]`https://github.com/B0tzki/DuplicateDetectorHierarchy`

3 Related Work

In this section, we review the related work. For that, we categorize the related studies into three groups: papers that address the duplicate question detector for SO posts, studies that focus on recommending SO tags, and works that refer to recommending Github tags.

3.1 Duplicate Question Detector for Stack Overflow

Since our study is focused on finding duplicate questions, we analyze studies that tackle this topic. For instance, the study by Ahasanuzzaman et al. [1] presented different ways of feature collection of a question pair and how they are combined with a Logistic Regression classifier for determining the duplicability of the question pairs. Similarly, based on this idea of determining the duplication probability of a SO post pair, the work by Zhang et al. [2] created multiple *similarity components*, combining them in a composer, which heuristically calculates the weights for each component. However, the research by Mizobuchi et al. [4] managed to improve those two models by extracting the *code snippet* and *strong* textual elements from the description of a question.

There are papers that used different techniques for creating the question embeddings besides the Word2Vec or TF-IDF approaches. For example, the paper by Pei et al. [17] proposed for question embeddings the Glove model [18] and for classification a deep learning technique. Another interesting idea is shown in the study by Zhang et al. [7] where for the vector similarity scores, the Doc2Vec model was used for generating the title and description embeddings. However, the ML classifier also considered the association score and topical similarity to spot the duplicate pair. Since, in reality, detecting a duplicate question is a highly computational operation, the work by Koswatte et al. [3] proposed a novel solution by using the hashing operation belonging to the *Semantic Text Similarity*.

Moreover, in attempting to improve the model performances on duplicate question detection, the scientist had adopted numerous Deep Learning techniques for the classification part, as they are shown in the papers [5, 6, 17, 19].

For our study, we consider the approach presented in work by Zhang et al. [2]; we split the problem of detecting the duplicate question into multiple components, namely: textual information score, tag score, and hierarchy score, and for computing the final score, we use as an automated solution a Logistic Regression algorithm instead of applying an heuristic approach. Concerning the model used for creating the textual information embeddings, we use the Doc2Vec model, shown in the paper by Zhang et al. [7]. The tag hierarchy represents the first novelty feature and is used to capture the semantical links between the questions. Moreover, it is important to mention that, because the dataset became very large and hard to do computations, in the studies [1, 3, 4, 5, 6, 7] the questions are split into categories based on their most common tags, i.e., Java, C++, Python, HTML, etc., while in our research we do not do this kind of split and we consider all the questions. In this manner, we can compute the performance and usefulness of adding the hierarchy and tag scores.

3.2 Tag recommendation for Stack Overflow

Because software engineers use the Stack Overflow website, scientists tackle the problem of recommending the tags for the SO posts. It is essential to note that each paper defines a Stack Overflow question differently. For example, while papers [8, 9, 10, 11] treated only the textual content from the description, others also consider the title [12] and code snippets [13, 14]. Additionally, it is also important to determine the best way of encoding

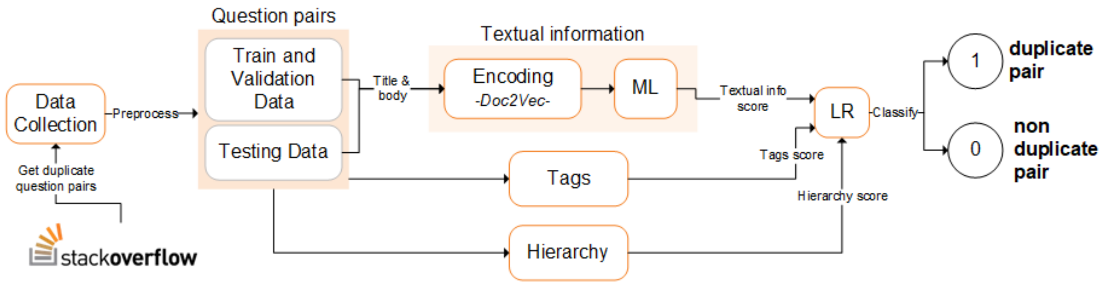

Figure 1: The pipeline used during the study

the textual information, either using the Word2Vec approach [20] as it is exploited in the works [12, 21, 22], or the TF-IDF technique presented in the research by Wang et al. [22].

In our work, we use the strategy presented in the study by Xu et al. [14] for building the textual information component. Essentially, we split the textual information of a SO post into three parts, i.e., title, body, and code snippet. However, we consider only the title and body encoded versions for creating the respective question embedding.

3.3 Tag recommendation for Github

There was a high interest in tagging recommendation problems in the Software Engineering domain. The issue can be synthesized in a more general way as given a Software Engineering entity, what are the most meaningful tags which correctly describe the software entity. The researchers started to define the software entities and tackle this problem from different perspectives. For instance, there is much literature on other recommender systems that assign suitable tags for a Github repository [15, 16, 21]. In the spirit of improving the performance of this kind of software, researchers found new features which can also be used for the other software entities. To be more concrete, Izadi et al. [15] proposed the KGRec and KGRec+ models, which besides the meticulous process of mapping the user tags to the featured Github tags, they use the information obtained from a knowledge graph, thus exploiting the semantic links between the tags.

Since we are interested in a method to capture the semantics between the questions and in this manner improving the duplication question detection system, we use the method presented in the research by Izadi et al. [15]. Concretely, we take the knowledge graph from this paper, build different tag hierachies and evaluate them to determine the duplicate question pair task. Additionally, we consider that this procedure can also be extended to recommending systems for similar SO questions or Github repositories.

4 Proposed Appproach

The SO question consists of *four* main elements - the post's title, body description, metadata, and list of tags. To solve the problem at hand, we build an ML-based model that receives a pair of SO posts as input and analyzes its textual information. Besides the textual information, we introduce a tags scoring component and an element of novelty that acts as a constraint factor and therefore improves the textual information score - a semantic-based hierarchy built based on the knowledge graph presented in the study by Izadi et al. [15].

To achieve the desired outcomes, we split the entire approach into multiple major parts, which can also be seen in Figure 1: collect the duplicate question pairs from SO, apply preprocessing operations, split the data into training, validation, and testing, build the textual information, tags, and hierarchy scoring methods and prepare the Logistic Regression model for the classification phase.

Concerning the dataset, we gathered the Stack Overflow duplicate question pairs on 24.05.2022 using the Stack Exchange API; we have 688,937 question pairs and 897,592 duplicate questions. The reason behind using this method is that we want to have the most recent data, and this can be illustrated by the fact that the last duplicate question found in our dataset was created on 22.05.2022. Besides that, the dataset also contains old duplicate questions, the first one being created on 31.07.2008. However, it is important to mention that we do not use all those questions - an additional explanation concerning why and how we filter them based on statistics is presented in Section 5.1. Thus, we resume our study with 10,000 question pairs and split them into training, validation, and testing datasets according to the 8:1:1 ratio. Furthermore, since we only collect positive samples and want to avoid bias, we randomly generate the same number of non-duplicate question pairs as the duplicate ones for each dataset - more details about the exact procedure are shown in the Section 5.1.

After this step, we use the questions' textual information from the training and validation set to train the embedding model. To gather the information effectively, we apply beforehand some textual preprocessing operations on the post's title and body, described in more detail in the Section 4.1. The main argument for utilizing the Doc2Vec model to obtain the corresponding question embeddings is that it can precisely catch the semantical meaning between the documents, comparing to Word2Vec or TF-IDF approaches. In Section 4.2, we describe how we obtain those question embeddings using the text from the title and body. Next, given the encoded version of the SO post pair, we create an ML-based model which outputs a probability of labeling each question pair as duplicate - close to 1.0 if the pair is a duplicate one, close to 0 otherwise. The output probability represents the textual information score. In Section 4.3, we present the ML-based models used in our study. To improve those numbers, we make the tags scoring component and more details about how it acts are shown in Section 4.4. In addition, to capture the semantics between the tags, we build a tag hierarchy and used it in a new scoring element - further details about them can be found in the Section 4.5 and Section 4.6, respectively. Finally, to find the most suitable weights for each corresponding component in an automated manner, we use a Logistic Regression algorithm that outputs the probabilities for the classification problem based on the input scores.

4.1 Preprocessing textual information

To determine if a pair of questions represents a duplicate one or not, we also consider the textual information from the title and the question description. To gain as much useful information as possible, we preprocess the texts. It is important to mention that the textual information contains some features in the original dataset that should be filtered. For instance, the description and the titles can contain HTML tags that should be removed. Moreover, the question body contains HTML tags which should be extracted and treated separately as another possible piece of information. Therefore, we split those HTML labels into two separate categories as they are also mentioned in the work by Mizobuchi et al. [4]:

- strong annotation: `<strong>...</strong>`, `<em>...</em>`, `<code>...</code>`
- code snippet annotations: `<pre><code>...</code></pre>`

Based on the aforementioned aspects, we do the following preprocessing steps for the question description:

1. Remove all the characters which are not ASCII (for instance, we can find symbols like π which can not be processed).
2. Remove all the markdown annotations.
3. Extract the code snippets and make a separate column for those.
4. Extract the strong annotations and make a separate column for those.
5. Remove the URLs and strings which might contain possible information about the duplicate questions (i.e., "Possible Duplicate")
6. Eliminate `\n` characters and lower all the characters.
7. Apply the mapping of keywords according to some heuristic rules[2].
8. Remove the HTML tags and handle the punctuation.
9. Tokenize the text, remove the stopping words and apply the Porter Stemmer algorithm described in work by Poter et al. [23]. For all those operations, we use the NLTK Python library [24].

Similarly, we apply the same steps for the preprocessing of the title, except for steps 3 and 4, since the title does not contain any code snippets or strong annotations.

4.2 Question Embeddings

We construct the question embeddings after preprocessing the titles' textual information and the questions' description. As a solution, we work with the Doc2Vec model to provide encodings both for the title and body text, and in the end, those are merged, building the question embedding.

For this task, we evaluate different Doc2Vec configurations for both title and body questions - we note them as $model_{title}$ and $model_{body}$, respectively. Furthermore, we use the embedded pairs of questions as a dataset for the ML-based model. Therefore, to tune the hyperparameters for both $model_{title}$ and $model_{body}$, we create all the possible combinations of the post embeddings and evaluate them on a default Logistic Regression model using the validation question pairs dataset. Finally, we select the top 3 best Doc2Vec combined models based on their accuracy, recall, F1-score, and precision performance. We focus on showing the results only on the best pair Doc2Vec configurations for conciseness.

[2]The entire list can be consulted here: `https://github.com/B0tzki/DuplicateDetectorHierarchy`

4.3 Textual information score

To generate the probabilities (scores) for classifying the textual information of a question embedding pair as a duplicate one or not, we analyze multiple ML-based models: Support Vector Machine (SVM), Logistic Regression (LR), Decision Tree (DT) and K-Nearest Neighbors (KNN). Moreover, to efficiently do the hypertunning parameter operation, we use for learning the training set of pairs, while for testing the validation dataset. We choose the best set of parameters for each model based on their accuracy, recall, F1-score, and precision performance - a list with them is described in Section 5.2. In addition, in Section 6, we utilize those models' configurations for reporting the textual information score along with tags' and hierarchies' scores.

We use the question embeddings pair generated by the Doc2Vec models discussed in Section 4.2 as input for each above-mentioned ML algorithms. To be consistent in learning and evaluation, we carefully set the following rule for all question pairs in all 3 types of dataset: for a pair of questions (q_i, q_j), its embedded version used as input is represented by the $(model_{title}(q_i) + model_{body}(q_i) + model_{title}(q_j) + model_{body}(q_j))$ if $i < j$, otherwise we swap the positions of the questions. In other words, we ensure that the question pairs are sorted in such a way that the *id* of the first question is smaller than the *id* of the second. Finally, the ML-based model returns a probability for labeling each question pair as a duplicate one. If the probability is greater than 0.5, the pair is labeled as 1 (duplicate pair) and 0 otherwise.

4.4 Tags score

Besides the textual information, we are also interested in considering the tags scores. To formalize the problem, we assume that given two sets of tags, this component should output the similarity probability between them. Thus, we apply the Jaccard Similarity formula on the tags set, as shown in Equation 1. This type of formula is preferred because, for this component, measuring the set's overlap represents the desired goal, neglecting the order of the tags in their sets.

$$Score_{tags}(q_i, q_j) = \frac{|tags_{q_i} \cap tags_{q_j}|}{|tags_{q_i} \cup tags_{q_j}|} \tag{1}$$

4.5 Building Tag Hierarchy

One of the most important aspects of this research is how we can capture the semantics between the tags using a hierarchy and if this type of data structure improves the behaviour of the duplicate question detection.

To create the final version of the hierarchy, we use multiple techniques to gather and combine the information from the knowledge graph described in the study by Izadi et al. [15]. One of them is represented by creating a fully automatic hierarchy based on the dendrogram from Agglomerative Clustering. We combine this approach and compare the results with the method of making an automatic cluster based on the Statistical Inference [25, 26], and Modularity [27, 28] techniques applied on the SED-KGraph [15] and adjust it for the hierarchy manually. After analyzing different clusters obtained by applying those heuristics, we conclude with three versions - h_full with 64 levels obtained using the Agglomerative clustering, h_stat with 5 levels obtained from Statistical Inference, and h_mod with 3 levels obtained from Modularity with the resolution value equal to 1.0. Since we consider that those hierarchies have some missing points, we also build another one manually: h_manual with 7 levels.

After this step, we consult the experts and document their conflicts - for the h_manual, we have 21 conflicts, while for the h_mod, we have 17. Furthermore, they suggest that most inconsistencies are due to generalizations. For instance, the h_mod contains less number of clusters on the lowest level, and we have some inaccuracies, while h_manual includes a large number of groups on the deepest level. Hence, they consider that the h_stat represents the most balanced cluster version, representing the middle way between them. Additionally, they observe that the h_full provides a very complex hierarchy, which can be used if we want to consider all the aspects between two tags when we cluster them.

4.6 Hierarchy Score

For computing the hierarchy score, we label each node in the hierarchy with a specific probability from 0 to 1. The idea is that the deeper nodes have a higher probability since if two tags can be found in the same node, they are likely to refer to the same domain. Thus, we start to label all the leaves with a score of 1.0 and continuously decrease it to 0 for the root. This labeling process was done manually for the h_mod, h_stat, and h_manual since experts analyze the domains to which the node refers and how large that node is. Nevertheless, for the h_full hierarchy, since there are a lot of nodes and in total there are 64 levels, we utilize an automatic labeling process in the following way: starting from the root of the hierarchy, we label the root with a score of 0 and continuously increase the score with a step of 0.015625 for each depth level.

To use this hierarchy component, we consider the following scenario: Given two sets of tags from two different questions, the hierarchy outputs their similar scores. We can reduce the problem to two tags from two different questions since, for the set case, we can apply the same procedure for each pair and, in the end, output the maximum score between those. Thus, if we have tag_m and tag_n, we can check the deepest level where they are in the same node and output the node's score.

5 Experiment Design

This section presents all the setup elements used for the experiments. For that, we split the experiment design into two categories: one which concerns information about the dataset, such as data collection, statistics, and dataset split, and the other one presents all the parameter values of different ML-based models used in the experiments.

5.1 Dataset

Regarding the dataset used in our research, we opt for the most up-to-date data from the Stack Exchange API, and therefore we collected the duplicate question pairs from Stack Overflow on 24.05.2022; we have 688,937 question pairs and 897,592 duplicate questions. We have some statistics and plotted their results in Figure 2. We observe that 17.26% of those questions do not have any tags in the SED-KGraph. Since our study also focuses on how a tag hierarchy can perform as a component of the duplicate detector system, we remove the questions that do not contain at least one tag in the hierarchy. Moreover, we observe that we do not have any questions which contain 0 tags, and most of them contain between 2-3 tags

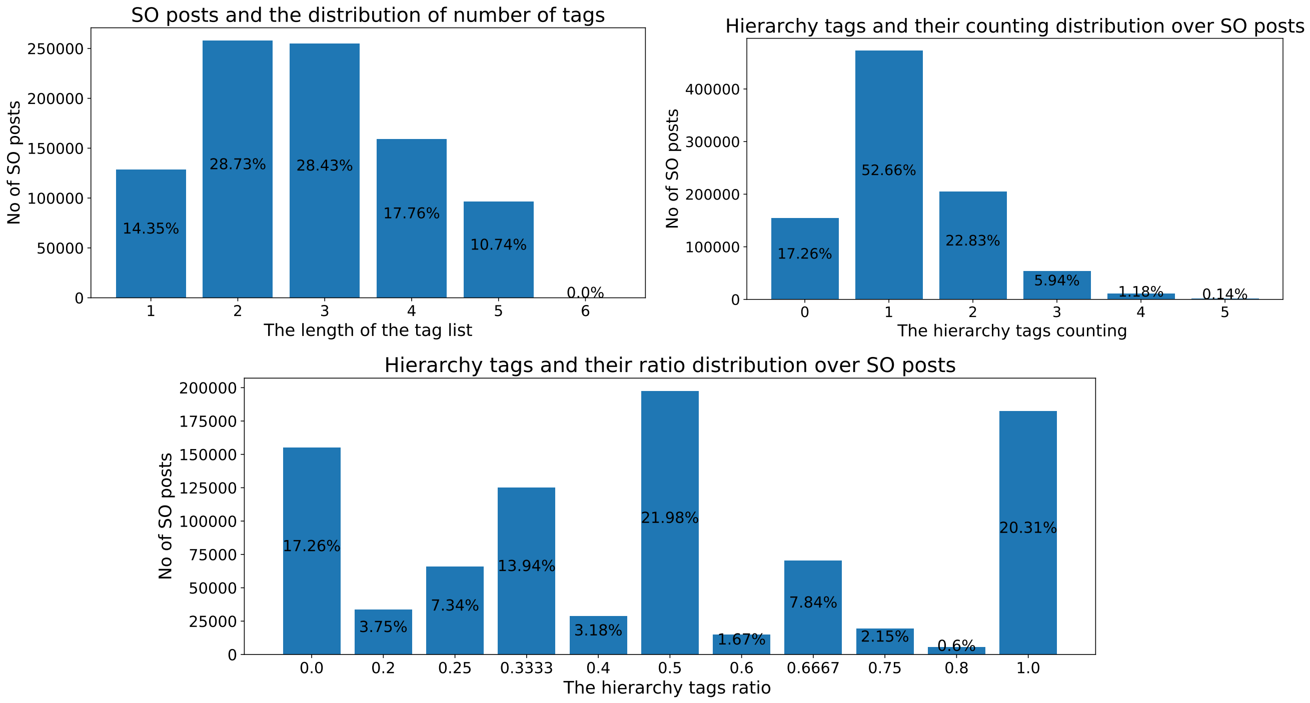

Figure 2: Statistics of the original dataset using percentage

Table 1: Statistics regarding the used dataset

Dataset	#positive instances	#total question pairs	# questions
Training	8,000	16,000	14,627
Validation	1,000	2,000	1,914
Testing	1,000	2,000	1,917

(approximative 57.16%). After removing the questions that do not satisfy the mentioned criteria, we obtain a dataset with 543,214 duplicate question pairs and 742,670 questions.

Forwards, due to the time constraint of the project and the limited computational power, we focus our study just on 10,000 duplicate question pairs. Thus, we randomly select this amount of question pairs from the original dataset, obtaining a final number of 18,075 questions. As a note, we consider the training, validation, and testing datasets based on the number of duplicate question pairs.

For the training, validation, and testing datasets, we split the obtained duplication question pairs based on the ratio of 8:1:1. Furthermore, to avoid having many questions in more than one dataset, we split them based on their appearance in the dataset. We obtain consistent numbers also shown in Table 1.

Since the original data does not contain any negative instances and to avoid the ML algorithm's bias, we also create non-duplicate question pairs for each type of dataset in the following manner: we generate the negative instances by randomly picking the questions from that dataset and checking if the generated pair is not already a positive instance. In

Table 2: Hyperparameters space for the Doc2Vec Title and Body Embeddings

Hyperparameters	Option values
Training Algorithm type for both Title and Body Embeddings	(Distributed Memory Model + Skip-Gram for word-vectors), (Distributed Bag of Words)
Dimension Title Embeddings	70, 100, 150, 200, 250, 300
Dimension Body Embeddings	400, 500, 600, 700, 800, 1000, 1100

Table 3: Hyperparameters and their values for the textual information classification models

Models	Hyperparameters and their values besides the default ones
SVM	kernel = 'poly', degree = 3
DT	max_depth=None, min_samples_leaf=2
KNN	n_neighbors=5, algorithm = 'kd_tree', weights="distance"
LR	C = 0.5, penalty = 'l1', tol = 1e-8, solver = 'saga'

the end, we use for the study: 8,000 duplicate and 8,000 non-duplicate question pairs for training, 1,000 duplicate and 1,000 non-duplicate question pairs for validation, and 1,000 duplicate and 1,000 non-duplicate question pairs for testing.

5.2 Configuration and implementation

In our study, for building the Doc2Vec model, we use the Python framework Gensim [29]. Additionally, for finding the best Doc2Vec models, besides the default framework values of the parameters, we set the **starting_learning_rate = 0.025**, **minimum_learning = 0.00025**, **min_count**[3] **= 3** and **epochs = 70**, for all types of the encoding models. The other hyperparameter values we explore can be found in Table 2.

Next, we use the ML-based approaches, namely the SVM, LR, DT, and KNN, to conduct experiments to search for the best configuration that can classify the question pair only by using its embedding. We utilize these model versions implemented in the Scikit-Learn Framework [30]. To keep the paper concise, besides the default values of the models, Table 3 presents the list of parameters and their values for which the best results are obtained.

Combining the textual information score with the hierarchy and tags score requires another ML-based approach to automatically compute the suitable weights for each component. Hence, we consider the Logistic Regression model from Scikit-Learn with its default parameters for this step.

Concerning the experimental hardware setup, we conduct our experiments on the Google Collab platform with a single core Intel(R) Xeon(R) CPU @ 2.20GHz, and 13 GB RAM.

[3]Ignores all words with total frequency lower than given value (Gensim documentation)

6 Results

We first want to address the configuration for the best Doc2Vec models based on the accuracy, recall, F1-score, and precision metrics. As seen in Table 4, besides the default values parameters discussed in Section 5.2, we report the results of the best hyperparameters values for the question embedding model. It is noted that we do not use the same training approach for both models. Concretely, for the $model_{title}$, we get the best results using the Distributed Memory approach combined with the Skip-Gram model for the word level. In contrast, the Distributed Bag of Words is preferred for the body embeddings. Regarding the size of the embeddings, the model seems to obtain more precise encodings for the titles using 70 features. At the same time, for the body embeddings, we use a dimension of 600 features to obtain accuracy and recall above 56.00%.

Next, we utilize the configurations mentioned earlier for generating the question encodings and afterward for training and evaluating the ML-based models for the classification process. Firstly, we analyze the results only from ML Classifiers and use the Gaussian NB, DT and KNN classifiers as baselines. Afterward, we combine the probabilities obtained from the SVM and LR configurations with the scores obtained from the tag and hierarchy components to see if their utilization can improve the duplicate question detection. As mentioned in Section 5.2, combining those probabilities to obtain a final score is done through a Logistic Regression algorithm. These final results are presented in Table 5. The extra column from the table shows the weights for each component; in this manner, we can deduce how much influence a feature has in computing the final scores.

As we notice, besides the LR model alone and LR + h_full, all the other configurations can deliver better results than the baselines regarding the accuracy score. We can also observe that the LR + h_full behaves similarly to the LR alone, an aspect also proved by the coefficient list; using the h_full for this model does not increase the performance, and only 0.43 of it is taken into consideration. Moreover, as in all cases, h_full does not offer information that can be used in the learning process, hence increasing the score. In most cases, the value of its weight is close to 0, which suggests the uselessness of this component. Additionally, in the case of LR + h_full + tags, the coefficient of the hierarchy usage receives a negative value, suggesting that the model is trying to adjust the score through the other two components.

On the other hand, we obtain the best accuracy results using the LR + h_manual + tags with a value of 92.10%, closely followed by the one that used the h_stat (92.05%). Even though the difference between those two setups is close, for the first setup, the hierarchy is used only for 0.23 while the h_stat from the second one is used more than the textual information analyzed by the LR (4.94 the coefficient for h_stat while for the LR is 4.49).

Next, even though we obtain the best textual information classifier score using the SVM model, we can see that the hierarchy and tags components still improve the setup performance, but much less than the case of the LR. Besides that, we do not manage to get for any of its setups an accuracy above 90%. Still, we can remark the best result obtained using the SVM + h_stat + tags (accuracy 89.5%), and the hierarchy component weighted closely the same amount as the tags one (4.24 the coefficient for h_stat while for the tags is 4.55). In all cases for this model, we can see that the main information is gathered from the probabilities obtained from the SVM classifier, having all coefficients above 12, while in the case of the LR, the main knowledge is obtained from the tag component.

Table 4: The best Question Embeddings Models based on the Logistic Regression results

Hyperparameters	Values Title Embedding		Values Body Embedding	
Model	Distributed Memory + Skip-Gram for word-vectors		Distributed Bag of Words	
Dimension Embeddings	70		600	
	Accuracy	Recall	F1-score	Precision
Logistic Regression score	56.70%	56.35%	59.40%	56.73%

7 Discussion

Overall, we observe that we canx improve the baselines score by using the SVM model. This is expected since the SVM using the polynomial kernel can better classify non-linear data as the question embeddings represent. Additionally, the tags and the hierarchy components increase the initial metrics. Still, we can not neglect that a bottleneck is created, and the accuracy, recall, and F1-score can not go beyond 90%. We think this happens because, in all cases, the most information for the classification process is obtained from the textual information component and not from the others. The SVM probability coefficient is above 12% in all cases, while the tags do not go further than 6.5, and for hierarchy, the maximum value is 4.24. Since, for this textual information setup, the accuracy and recall are already high (accuracy above 83% and recall above 77%), the improvement is not a considerable one in comparison with the LR model - for the best configuration, the accuracy increases by +6.55%, and the recall by +10.13%.

For the LR model, we get a considerable improvement using the hierarchy and tags components - for the best configuration, the accuracy increase by +68.83%, and the recall by +68.87% - compared with the LR model. We assume that we obtain this progress percentage for the LR model since, in reality, the textual information probability from LR has a very small influence; its coefficient does not go above 4.70, while the tags are mainly used with a minimum coefficient of 11.95. In general, when only the hierarchy and the textual information are used, the hierarchy plays a more important role than the LR results probabilities; the only exception is for h_full.

Regarding the hierarchy components, we can see a big gap between how the h_full acts compared to the others. We consider that this happens because the h_full is the most detailed one. Therefore, we should use a hierarchy containing more general groups to classify the questions. A good example in this regard is illustrated by the h_stats and h_manual, both representing the middle way between the h_mod and h_full; the h_mod contains 3 levels, while h_stat has 5 and h_manual has 7. Moreover, it is also important that we should not use a small hierarchy as h_mod since, with many general labels, for the hierarchy component becomes very challenging to output the probability duplication score. However, suppose the hierarchy components are used for this kind or similar investigation, then a hierarchy that is not very deep, with levels between 5 and 10, should be more than enough to give close probabilities to classifying the question pairs and to determine the semantic links between the tags.

Lastly, an important observation is that analyzing the tags is very useful based on the results obtained. We think this approach works well with the SO questions since the tags used for this website are more detailed than those used in Github. In addition, there are

Table 5: Results Classification models

Configurations	Accuracy	Recall	F1-score	Precision	Coefficients
Gaussian NB	52.35%	52.73%	52.58%	45.30%	
DT Classifier text	53.4%	53.09%	53.51%	58.3%	
KNN Classifier text	56.95%	54.26%	61.7%	88.5%	
SVM text	83.0%	77.59%	83.16%	92.8%	
SVM text + tags	88.44%	84.05%	88.49%	94.9%	[13.2, 6.34]
SVM text + h_mod	85.25%	80.41%	85.34%	93.2%	[14.2, 3.29]
SVM text + h_mod + tags	88.85%	84.41%	88.89%	**95.3%**	[13.11, 1.63, 5.67]
SVM text + h_stat	87.45%	83.22%	87.5%	93.8%	[13.55, 6.87]
SVM text + h_stat + tags	**89.5%**	**85.45%**	**89.53%**	95.19%	[12.95, 4.24, 4.55]
SVM text + h_manual	83.85%	78.61%	83.98%	93.0%	[14.56, 2.07]
SVM text + h_manual + tags	88.35%	83.9%	88.4%	94.9%	[13.2, 0.54, 6.21]
SVM text + h_full	83.1%	77.72%	83.26%	92.8%	[14.77, 0.91]
SVM text + h_full + tags	88.35%	83.9%	88.4%	94.9%	[13.22, 0.37, 6.32]
LR text	54.55%	54.29%	54.58%	57.49%	
LR text + tags	92.0%	91.66%	92.0%	92.4%	[4.49, 16.04]
LR text + h_mod	76.09%	71.25%	76.41%	87.5%	[4.61, 6.24]
LR text + h_mod + tags	92.0%	91.17%	92.0%	93.0%	[4.48, 1.47, 14.63]
LR text + h_stat	88.3%	83.47%	88.36%	**95.5%**	[4.48, 13.74]
LR text + h_stat + tags	92.05%	90.31%	92.05%	94.19%	[4.49, 4.94, 11.95]
LR text + h_manual	65.4%	63.02%	65.68%	74.5%	[4.65, 4.75]
LR text + h_manual + tags	**92.1%**	**91.68%**	**92.1%**	92.6%	[4.49, 0.23, 15.89]
LR text + h_full	53.9%	53.65%	53.95%	57.2%	[4.73, 0.43]
LR text + h_full + tags	92.0%	91.66%	92.0%	92.4%	[4.47, -0.87, 16.16]

more default tags in SO than in Github (approximative 63,256 SO tags), which contributes to the multiple and detailed options that users can utilize for labeling their posts better.

8 Threats to the Validity

The threats to the validity are presented in this section. Based on study [31], we divide them into three validity categories: internal, external, and construction.

8.1 Internal Validity

For the internal validity, we present the threats which might influence the final results due to the manual operations done by the experts. Namely, in our study, we used the expert's knowledge to adjust some semantic-based tag hierarchies (h_mod, h_stat, h_manual). Even though we conducted experiments to track their performances, we can not ensure that the presented versions of these hierarchies are the best and do not exist others that behave better in a real environment. Moreover, it is also important to mention that we obtained good results with these hierarchies only in the setup and dataset presented in the paper.

8.2 External validity

External validity is associated with the generalizability of the results. The instances from the used dataset represent the first threat in this category. As mentioned in the Section 5.1, we took the duplicate question pairs from the SO database. Unfortunately, due to the time constraints and the number of question pairs that have to be analyzed, we did not check them manually to see if they are missed classified or not, and we trust the labels from the SO database. Thus, there might be a risk that we used for the learning and testing process pairs that might be misclassified.

Another threat we consider is represented by the size of the dataset we used for this study. As mentioned in the Section 5.1, we took 10.000 random pairs from the original dataset. Hence, it is important to mention that since we did not work with the original size dataset provided by the Stack Overflow, we can not conclude how the built system behaves in a real environment where much data is used. Besides that, using the noted split of the dataset, our model can not consider topic trends (Stack Overflow provide plots regarding this aspect for each used tag), which might affect its performance in an online environment.

Lastly, we create the dataset setup also based on the topics seen in the SED-KGraph. Therefore, even though the knowledge graph is built for the Github repository tags and not for SO tags, we filtered the questions and consider only the ones with at least one tag in the knowledge graph. However, using the SED-KGraph can limit the performance of the ML-based model in a real environment since the SO tags are more than the Github tags.

8.3 Construction Validity

This part refers to the theoretical metrics used to assess our study. During the experiments, we used standard theoretical metrics such as accuracy, recall, F1-score, and precision. Moreover, in all types of datasets (training, validation, and testing), we carefully avoided biases by having a 1:1 ratio between positive and negative samples.

9 Conclusions and Future Work

In conclusion, we can affirm that using a suitable hierarchy can increase the performance in finding duplicate questions besides the textual information. Nevertheless, we observe that using a very deep hierarchy does not help the system achieve better predictions, and in most cases, this kind of hierarchy acts as a bottleneck. Moreover, a hierarchy containing a small number of levels can not produce good prediction results. In our study, we obtain the best improvement percentage using hierarchies that had between 5-10 levels, and we consider this a good heuristic in terms of how detailed a hierarchy should be. This remark can also be used in future works that tackle similar research questions.

Moreover, the tag component also adds significantly enhances solving this classification problem. Therefore, we consider that this component should be used especially for the SO duplicate question pairs detectors since SO provides a large library of tags (approximative 63,256).

In terms of future work, we think that there can be done an extensive study to determine which embeddings models are more suitable for this kind of task. For example, our study uses the Doc2Vec model to generate title and body embeddings. However, there is a gap to give a try for an average Word2Vec, TF-IDF approach, pre-trained word embeddings models such as Glove, or if there is enough time and computational power, researchers can go further and use different types of transformers.

Additionally, there is a possibility to test the Deep Learning models to obtain the best configuration for classifying the encoded version of the pair. Moreover, it is important to mention that our study uses textual information from title and body. However, there is also a possibility to obtain embeddings from the code snippets or strong components. We do not use those elements in our work because we have many questions in the dataset that do not contain those two features, but this setup can be explored in future investigations.

The type of the hierarchy represents another important aspect. For this study, we use the SED-KGraph from the study by Izadi et al. [15], which was explicitly constructed for Github tags. Even though we obtain promising results with it, building a hierarchy based on the SO tags might be useful and improve the models' performances more.

Lastly, it is noted that due to the time and computational power constraints, we use a total of 10,000 question pairs instead of using the entire dataset. We consider that utilizing the entire data can affect the system's performance and provide a better overview of this topic.

References

[1] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "Mining duplicate questions of stack overflow," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pp. 402–412, IEEE, 2016.

[2] Y. Zhang, D. Lo, X. Xia, and J.-L. Sun, "Multi-factor duplicate question detection in stack overflow," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 981–997, 2015.

[3] D. D. Koswatte and S. Hettiarachchi, "Optimized duplicate question detection in programming community q&a platforms using semantic hashing," in *2021 10th International Conference on Information and Automation for Sustainability (ICIAfS)*, pp. 375–380, IEEE, 2021.

[4] Y. Mizobuchi and K. Takayama, "Two improvements to detect duplicates in stack overflow," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 563–564, IEEE, 2017.

[5] L. Wang, L. Zhang, and J. Jiang, "Detecting duplicate questions in stack overflow via deep learning approaches," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 506–513, IEEE, 2019.

[6] L. Wang, L. Zhang, and J. Jiang, "Duplicate question detection with deep learning in stack overflow," *IEEE Access*, vol. 8, pp. 25964–25975, 2020.

[7] W. E. Zhang, Q. Z. Sheng, J. H. Lau, and E. Abebe, "Detecting duplicate posts in programming qa communities via latent semantics and association rules," in *Proceedings of the 26th International Conference on World Wide Web*, pp. 1221–1229, 2017.

[8] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "Entagrec++: An enhanced tag recommendation system for software information sites," *Empirical Software Engineering*, vol. 23, no. 2, pp. 800–832, 2018.

[9] X.-Y. Wang, X. Xia, and D. Lo, "Tagcombine: Recommending tags to contents in software information sites," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 1017–1035, 2015.

[10] P. Zhou, J. Liu, X. Liu, Z. Yang, and J. Grundy, "Is deep learning better than traditional approaches in tag recommendation for software information sites?," *Information and software technology*, vol. 109, pp. 1–13, 2019.

[11] P. Zhou, J. Liu, Z. Yang, and G. Zhou, "Scalable tag recommendation for software information sites," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 272–282, IEEE, 2017.

[12] S. K. Maity, A. Panigrahi, S. Ghosh, A. Banerjee, P. Goyal, and A. Mukherjee, "Deeptagrec: A content-cum-user based tag recommendation framework for stack overflow," in *European Conference on Information Retrieval*, pp. 125–131, Springer, 2019.

[13] J. He, B. Xu, Z. Yang, D. Han, C. Yang, and D. Lo, "Ptm4tag: Sharpening tag recommendation of stack overflow posts with pre-trained models," *arXiv preprint arXiv:2203.10965*, 2022.

[14] B. Xu, T. Hoang, A. Sharma, C. Yang, X. Xia, and D. Lo, "Post2vec: Learning distributed representations of stack overflow posts," *IEEE Transactions on Software Engineering*, 2021.

[15] M. Izadi, M. Nejati, and A. Heydarnoori, "Semantically-enhanced topic recommendation system for software projects," *arXiv preprint arXiv:2206.00085*, 2022.

[16] T. Wang, H. Wang, G. Yin, C. X. Ling, X. Li, and P. Zou, "Tag recommendation for open source software," *Frontiers of Computer Science*, vol. 8, no. 1, pp. 69–82, 2014.

[17] J. Pei, Y. Wu, Z. Qin, Y. Cong, and J. Guan, "Attention-based model for predicting question relatedness on stack overflow," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pp. 97–107, IEEE, 2021.

[18] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[19] Z. Liao, W. Li, Y. Zhang, and S. Yu, "Detecting duplicate questions in stack overflow via semantic and relevance approaches," in *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 111–119, IEEE, 2021.

[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[21] M. Izadi, A. Heydarnoori, and G. Gousios, "Topic recommendation for software repositories using multi-label classification algorithms," *Empirical Software Engineering*, vol. 26, no. 5, pp. 1–33, 2021.

[22] H. Wang, B. Wang, C. Li, L. Xu, J. He, and M. Yang, "Sotagrec: a combined tag recommendation approach for stack overflow," in *Proceedings of the 2019 4th International Conference on Mathematics and Artificial Intelligence*, pp. 146–152, 2019.

[23] M. F. Porter, “An algorithm for suffix stripping,” *Program*, 1980.

[24] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[25] T. P. Peixoto, “Bayesian stochastic blockmodeling,” *Advances in network clustering and blockmodeling*, pp. 289–332, 2019.

[26] L. Zhang and T. P. Peixoto, “Statistical inference of assortative community structures,” *Physical Review Research*, vol. 2, no. 4, p. 043271, 2020.

[27] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[28] R. Lambiotte, J.-C. Delvenne, and M. Barahona, “Laplacian dynamics and multiscale modular structure in networks,” *arXiv preprint arXiv:0812.1770*, 2008.

[29] R. Rehurek and P. Sojka, “Gensim–python framework for vector space modelling,” *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, vol. 3, no. 2, 2011.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[31] R. Feldt and A. Magazinius, “Validity threats in empirical software engineering research-an initial survey.,” in *Seke*, pp. 374–379, 2010.