Improving GitHub Tag Recommender Systems Using Tag Hierarchies

Arend van der Rande
Supervisor(s): Dr. Maliheh Izadi, Prof. dr. Arie van Deursen
EEMCS, Delft University of Technology, The Netherlands
22-6-2022

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

Programmers and software engineers often share code and one of the largest platforms on which this happens is GitHub, with an 87,58% market share in the Source Code Management Category [1]. One important part of sharing code is making sure that others who might be interested in it are also able to find it. One way to do that is by adding tags to a repository, which is a feature on GitHub only since 2017. However, many repositories do not have any assigned tags [2]. This can be solved by automatically applying tags to repositories without any. This comes with a problem: what algorithm can complete such a task?

In this study, we attempt to solve this problem using a class of algorithms called Hierarchical Multilabel Classifiers(HMCs). As the name suggests, these are a kind of classifier that can assign multiple labels to one datapoint (repository), but the labels must be organized in a hierarchy. We present 4 different hierarchies and 4 different HMCs to see which combination yields the best results. These combinations are also compared to a non-hierarchical baseline. We find that HMCN-F, one of the HMCs, manages to marginally outperform the baseline with a difference in AUPRC scores of 0,024. While not a groundbreaking result, it is promising, as other methods of creating hierarchies may be able to beat the baseline by a larger margin.

1 Introduction

Whether it is when answering questions on StackOverflow or contributing to a public repository on Github, software developers often share code. These are but two well-known code-sharing platforms and there are many more. Both of these platforms have a very large amount of questions and repositories, respectively. This poses a problem: how can a software developer find the type of repository they are looking for? The solution that is researched in this paper is to add tags to repositories. These tags convey relevant information for finding desirable repositories, such as what programming language is used if it uses machine learning, or what license it uses.

Since 2017 GitHub allows users to attach these tags to their repositories. Ideally, every repository has tags attached to it, as it would allow a searching software developer to find more relevant repositories. Currently, this is not the case: "as of February 2020, only 5% of public repositories in GitHub had at least one topic assigned to them" [2] (p. 93). A solution would be to automatically suggest tags for repositories that do not have enough tags to support the search algorithm or have no tags at all. Multiple studies have already been done that attempt to recommend tags for repositories based on their contents and metadata such as the studies were done by Izadi et al. [2] and Zhang et al. [3].

In this study, we consider a different approach. A hierarchical structure for these tags is developed, which can then be used for training Hierarchical Multilabel Classifiers (HMCs). HMCs have seen much development in the past 20 years. One of the reasons for this is the rapid development of artificial neural networks, caused by the introduction of powerful GPU architectures [4]. The aim of this work then is to experiment with different HMCs, including the current state-of-the-art developed by Giunchiglia et al. [5], and see if these provide an improved recommender system for tags on GitHub repositories compared to baselines from previous works [2], [3].

Our results show that HMCs can have similar performance to that of the baseline. With HMCN-F by Wehrmann et al. [6] we were able to achieve an AUPRC of 0,570 compared to the baseline of 0,556. The hierarchy for this achievement was built using agglomerative clustering between the labels with distance defined using the SED K-Graph, developed by Izadi et al. [7].

2 Problem Definition

The problem we aim to solve is defined as follows: we must find a hierarchy that can be used with an HMC to recommend tags for repositories, improving upon the baseline. To be exact: GitHub hosts a set of public repositories $S = \{r_1, r_2, ..., r_n\}$, with n being the amount of repositories and r_i being a repository. Each of these repositories always has an author and a title. They may have a README file, a description, wiki pages, and an arbitrary amount of files, most often some kind of programming code. GitHub also allows users to assign tags to their repositories. In theory, users can define any text as a tag. Users can also attach any string of text to their repository as a tag. In this research we consider a subset of these tags $T = \{t_1, t_2, ..., t_m\}$ where m is the amount of tags and t_i is a tag. This set does not contain all tags on GitHub but could be extended to an arbitrary amount. Using an algorithm, we order these tags into a hierarchy H, which we limit to a tree in this research. The wider class, DAGs, can be used for some HMCs such as AWX [4]. Then, a recommender system is built with the set of repositories S and the hierarchy H. This system can decide for each repository which of the tags from T are most relevant to assign to it. What is relevant is learned from existing relations between tags, repository information, and H.

3 Related Work

The related work is divided into 3 parts. The first part is about works regarding Hierachical Multilabel Classifiers and the second part is about approaches for applying tags to software entities. The final part outlines the connection this paper makes between the other two.

3.1 Hierachical Multilabel Classifiers

Early HMCs

In 2004, Lewis et al. developed an important implementation of an HMC. It became a reference point as state-of-the-art for other HMCs in the years to follow and is called Clus-HMC [8]. It is based on a concept called Predictive Clustering Trees (PCT). In 2010, an improved version of Clus-HMC was developed by Schietgat et al. [9]. They used the

Bagging technique [10] to train multiple, different classifiers using replication on the training set.

Another type of method was developed by Bi et al. in 2011, called CSSA [11]. It isn't restricted to tree-like hierarchies as DAGs are also allowed. It uses a greedy algorithm to traverse the tree and decide whether to assign nodes or not. It managed to provide a small improvement over Clus-HMC when trained on genetic datasets.

HMC-LMLP and HMCN

In 2014 another approach was taken by Cerri et al. called HMC-LMLP [12], where LMLP stands for Local Multi-Layer Perceptron. This is detailed, as it explains how it works: for locally classifying each level of the hierarchy, it uses a Multi-Layer Perception, which is a type of artificial neural network. The approach goes from top to bottom, where the classifier first learns to classify the high-level tree nodes using an MLP. Next, it learns to classify the second-highest level tree nodes from the predictions made by the first classifier. It continues to cascade down like this until it reaches the leaf nodes. After Clus-HMC, it managed to establish itself as state-of-the-art. Through the years it was improved upon, with HMCN as a significant result. It was developed by Wehrmann et al. in 2018 and technically it has two variants: HMCN-F and HMCN-R [6]. The difference is that they use a feed-forward and recurrent architecture, respectively. This has significant implications for the algorithm design. The significant difference compared to HMC-LMLP is that the input vector is used at all levels of the hierarchy, as well as the addition of a global loss function, calculated from the output from all levels of the hierarchy.

AWX

Using yet another approach is AWX. It was developed by Masera et al. in 2018 [4]. The classifier is described as a layer in a neural network. To adhere to the hierarchical constraints, they developed a loss function ensuring that the neural network's output is always a subtree of the full hierarchy.

C-HMCNN(h)

The last HMC we discuss here is C-HMCNN(h) from 2020 by Giunchiglia et al. [5]. It is the most recent major improvement in the field. It is somewhat similar to AWX, as it provides layers for a neural network. However, C-HMCNN(h) provides two layers instead of just one. The first layer is a constraint layer, which ensures that the hierarchy is not violated. The next layer is a loss function, which aids in teaching the classifier when to use information from the lower levels.

3.2 Applying Tags to Software Entities

Two papers are considered here regarding tag application. First, the paper by Zhang et al. from 2019 [3] introduces HiGitClass, and the paper by Izadi et al. from 2021 introduces a non-hierarchical tag recommender.

HiGitClass was developed to deal with three issues the authors encountered in recommending tags to repositories, which are " (1) the presence of multi-modal signals; (2) supervision scarcity and bias; (3) supervision format mismatch." [3] (p. 1). They developed a three-part framework that aims to tackle each of these problems. However, they experimented only on a specific subset of the data with relatively small hierarchies. A claim is made that their algorithm beats the other algorithms with a significant margin, but there are no comparisons made with the current or previous state-of-the-art HMCs.

The recommender introduced by Izadi et al. lays a lot of useful groundwork from which experiments can be set up for recommending topics to GitHub repositories. The data collection, tag mapping, and evaluation methods are all useful for experimenting and comparing different algorithms.

3.3 This Study

This paper is an attempt to combine the two previous sections: can we use HMCs for applying tags to software entities? The answer to this question is obviously yes, but we must also find out how this class of recommenders compares to the existing recommenders.

4 Background

The next step then is to understand what the current state-of-the-art classifier is. We also look at other classifiers, as it is very well possible that one of the other very good classifiers works better for this particular data set. From the classifiers detailed in the related works section, four are chosen for further experimentation. These are HMC-LMLP, AWX, HMCN, and C-HMCNN(h). The following subsections provide further detail, such as how they work and why they were chosen for this experiment. In the end, the baseline is also discussed. An overview is provided in Table 1.

From this table, we can see that there is high relative performance for C-HMCNN(h) compared to the previous state-of-the-art, HMC-LMLP. AWX and HMCN do both provide an improvement over HMC-LMLP, but if the goal is to find the best method, C-HMCNN(h) can be expected to perform well. The following subsections provide additional background information for each of the algorithms.

4.1 C-HMCNN(h)

C-HMCNN(h) is part of a neural network, with a special loss function that enforces the hierarchical constraints. Giunchiglia et al. also provided a Python implementation [5][1] with an example, which made it relatively easy to adopt for our experiments. The hierarchy itself needs to be represented as a matrix for the loss function, where given n classes it is a $n \times n$ matrix H where $H_{ij} = 1$ if class i is an ancestor of class j. This means that on column j there are multiple values equal to 1, as all classes have multiple ancestors, except for the highest level nodes. An example of such a matrix is visualized in Figure 1.

From this graph we can conclude a few things: by definition, the root with index 0 is the ancestor of all items and every node is its own ancestor. We know that there are $1 + 20 + 60 + 130 = 211$ non-leaf nodes, which we can also see in the figure: on the y-axis, we see that above $y = 210$ (indexing starts at 0) every node has five ancestors, which corresponds with the definition.

[1]https://github.com/EGiunchiglia/C-HMCNN

Name	Year	Description	Performance
C-HMCNN(h) [5]	2020	Comprised of two layers: the first is a constraint layer to enforce the hierarchy and the second is a loss function, which teaches the classifier when to use information from lower levels in the hierarchy.	1.238*HMC-LMLP, 1.123*Clus-Ens, 1.032*HMCNR
HMCN [6]	2018	Uses an artificial neural network in each of the hierarchical layers. Feedback propagates locally and globally. Can use either a feed-forward or recurrent architecture.	1.465*Clus-HMC, 1.340*CSSA, 1.110*Clus-ENS, 1.217*HMC-LMLP
AWX [4]	2018	Creates an output layer for an artificial neural network which is a subtree containing the tags we want to recommend.	1.191*Clus-HMC, 1.108*HMC-LMLP
HMC-LMLP [12]	2014	Uses an artificial neural network in each of the hierarchical layers, where each network is only given the output of the higher layer as input. The difference from HMCN is the lack of a global function.	1.075*Clus-HMC, 1.667*Clus-HSC, 1.745*Clus-SC

Table 1: Comparison of four Hierarchical Multilabel Classifiers, with performance calculated as relative AUPCR scores on the CellCycle dataset.

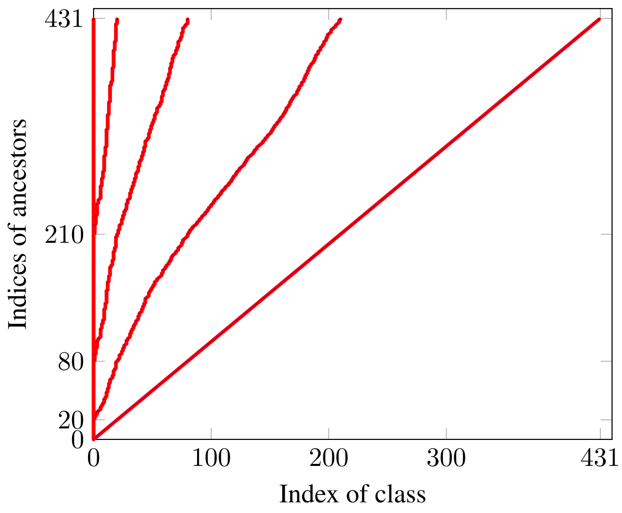

Figure 1: Ancestry matrix for hierarchy using the SED-K Graph relations and Bisecting K-Means clustering

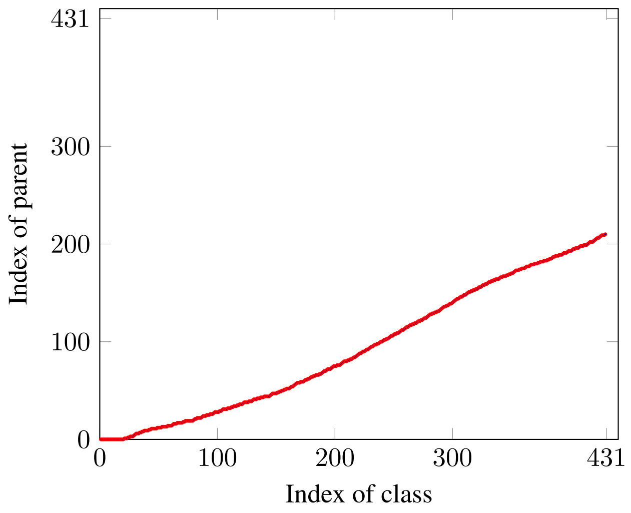

Figure 2: Parent matrix for hierarchy using the SED-K Graph relations and Bisecting K-Means clustering

4.2 HMCN

HMCN has two variants, HMCN-F and HMCN-R. In this paper, we focus only on the HMCN-F variant. It uses a neural network for each layer of the hierarchy, all of which use the output of the previous layer and the repository features as input. The last layer is a classifier for all the available classes, including the abstract layers. The output of each layer is also used as input in a global classifier. The final output of the local classifiers and that of the global classifier is combined at the end. This way, the classifier uses both the local and global information of the hierarchy. This is further explained in the paper by Wehrmann et al. [6].

4.3 AWX

AWX is also part of a neural network and uses the hierarchical information for the loss function, in a quite similar manner compared to C-HMCNN(h). An implementation is provided by the researches [4][2]. They also provide references to datasets that can work with their system, such as genetic data, which helps with figuring out how exactly it works. It uses a matrix to represent the hierarchy, but this one is different from the one used in C-HMCNN(h). Instead of giving all the ancestors for a node, it gives the index of the parent. More formally: it is a matrix H such that $H_{ij} = 1$ if j is the parent of i. This results in a curve such as in figure 2.

4.4 HMC-LMLP

HMC-LMLP is relatively simple compared to the other HMCs. All that is needed is a neural network between each of the hierarchical layers: four in total in this case. The sklearn library is used [13] for building, training, and predicting. The structure of the model is presented in Table 2. Each layer of the structure has an input layer, a hidden layer, and an output layer. The input layer of the first layer is the set of input

[2]https://github.com/lucamasera/AWX

features and the output of the last layer is the desired set of tags.

Training this model goes from top to bottom. The first layer is trained with the input features as input. The target data is a list l of length 20 where $l_i = 1$ if one of the leaves under the i-th subtree is a true label and 0 otherwise. When that layer is done training, it is used to make high-level abstract predictions based on the input features. These predictions are used as input data for the second layer. The target data is a similar list as the previous one, but now of length 60. This model is also trained and its predictions are then used as input for the next model. This process is repeated until we end up at the bottom, where we end up with a probability for each tag of whether or not it should be recommended.

5 Proposed Approach

The solution for recommending tags to repositories using an HMC consists of two parts: first, a hierarchy must be constructed. Second, an HMC must be created which can work with the hierarchy, be trained, and finally be used to create predictions for new data.

For the hierarchy, the reason and logic behind why a hierarchy makes sense in this context are given. Then, the hierarchies are constructed using a distance metric and a clustering algorithm. As there are two different distance metrics and clustering algorithms this results in 4 different hierarchies.

The HMC is a type of classifier that needs a hierarchy in the output classes, which it uses to its advantage. As explained in the Related Work section, 4 different HMCs are used in this experiment: AWX, C-HMCNN(h), HMC-LMLP, and HMCN-F. These are all trained using public data from GitHub repositories and the tags which are currently assigned to these repositories. This public dataset was preprocessed as described by Izadi et al. [2] and then converted to a feature vector using a Tf-Idf vectorizer, for which we used the sklearn implementation[13]. To evaluate each classifier we use a portion of the GitHub data which is set aside for testing and compare the predictions the model makes to the actual tags.

5.1 Constructing a Tag Hierarchy

The idea of ordering these tags in a hierarchy may seem arbitrary, but there are some contexts in which it makes sense. Take for example the tags ”programming-language”, ”java” and ”python”. It would make sense to say that both ”java” and ”python” are children of ”programming-language”, as these

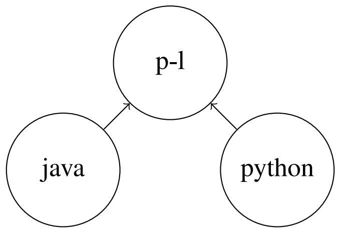

Figure 3: A simple example of a hierarchy, with programming-language abbreviated to p-l

are examples or instances. This is illustrated in Figure 3. This makes sense for this example, but there are two problems:

- Generalizing this concept is hard, as not all tags have a clear parent instance, whether it be a superclass, abstraction, or generalization.
- Attempting to apply hierarchical concepts to all 220 tags in our dataset by hand makes it harder to extend to a context where there are more tags.

So the idea of creating a hierarchy might make sense, at least in some cases, but a method is needed to do this automatically. The approach we use for this is clustering algorithms which have a hierarchical output. For such a clustering algorithm a distance metric between the tags is needed.

Distance Between Tags It is essential for the clustering algorithms that the distance between tags can be calculated. The problem with this is that on their own, the tags don’t have any context or relation to another. Nothing about the word ”java” says that it should be placed lower in a hierarchy of tags than ”programming-languages”. We identified four sources of context for these tags: descriptions of these tags from GitHub[3], Wikipedia articles, the SED-KGraph developed by Izadi et al. [7], which defines relations between tags and finally, the co-occurrence of tags in repositories. The descriptions from GitHub seem too short to be used for calculating the distance between the tags. Wikipedia articles suffer from the issue that there are a lot of tags related to specific libraries or frameworks for which there is no Wikipedia page, such as the CSS framework ”tailwind”. The SED-KGraph already has information about relations between tags, but not directly regarding hierarchy. Using the co-occurrence of tags in repositories should provide a hierarchy where tags are closer together in the hierarchy if they occur more often. From these four options, the SED-KGraph and the co-occurrence matrix were chosen to be used in the rest of the study.

Clustering Algorithms In order to build a hierarchy from these tags, a clustering algorithm able to provide a hierarchy is needed. Looking at a list of options, such as from sklearn[4], there seem to be two clustering algorithms with desirable properties: agglomerative clustering and bisecting K-Means. They both work with large amounts of samples and clusters. More importantly: building a hierarchy from these algorithms is a part of the algorithm. Agglomerative clustering (AC) provides a dendrogram, which can be converted to a hierarchy by cutting at certain points. These points can be chosen such that they give us the desired amount of nodes at a certain level. Bisecting K-Means (BK) follows a top-down approach: in each step of the algorithm, a cluster is divided. Thus, if we want a hierarchy with 10 layers at the highest level and 50 below that, we can divide until we get 10 layers and use that for the high-level hierarchy. Then, we can continue dividing clusters until we end up at 50 clusters and use that for the second-highest level cluster. In that case, it is guaranteed that if two tags are in the same lower-level cluster,

[3]https://github.com/github/explore/tree/main/topics

[4]https://scikit-learn.org/stable/modules/clustering.html

Layer	Input layer size	Hidden layer size	Output layer size
1	#Input features	#Input features	20
2	20	60	60
3	60	130	130
4	130	220	220

Table 2: Sizes of the various layers of the HMC-LMLP classifier

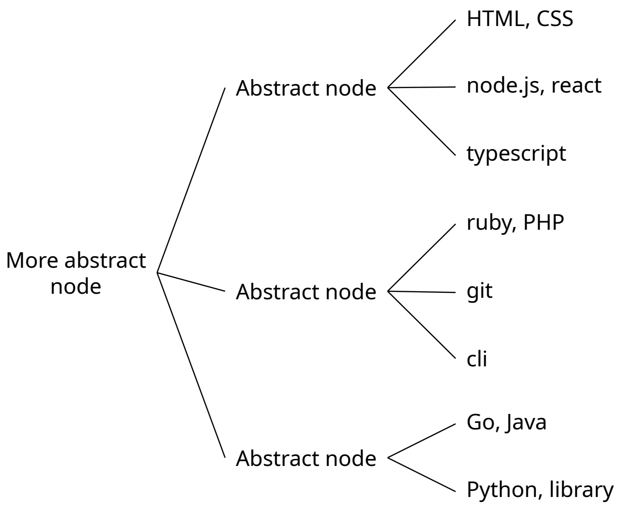

Figure 4: A subtree of a hierarchy constructed using the Co-Occurrence Matrix and Agglomerative Clustering

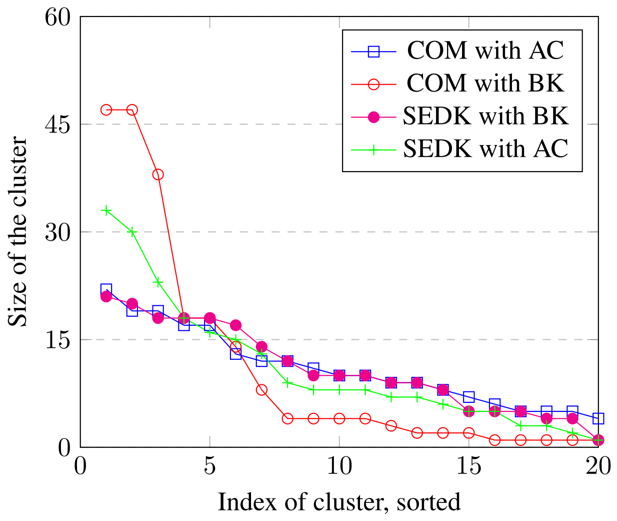

Figure 5: The cluster size for different algorithms at the highest level

they are also in the same higher-level cluster. Now, these two algorithms and datasets can be used to generate hierarchies. However, before we use them for training the classifiers, a short analysis of the size and content is performed.

Analysis The trees generated from this algorithm are too large to fit in this paper, but there is space for a part of the tree, which can be found in Figure 4. Especially the top abstract node seems useful, as it contains many web-development related languages. However, some additional insight into their construction is useful, as it allows us to reason why these trees might have different performances when used for classification. One such interesting property is the size of the clusters at each level of the hierarchy. While at this moment it is hard to say what exactly makes a good hierarchy for classification, having one cluster be significantly larger compared to the other clusters is assumed to have a negative influence on the performance.

As can be seen in Figure 5, most methods provide a decent cluster size spread. Combining the co-occurrence matrix with bisecting K-means seems to be the most imbalanced, but not drastically. Using the co-occurrence matrix with bisecting K-means provides two rather large clusters, which is reflected in the rest of the curve. What we can conclude is that these methods aren't skewed to a certain cluster, which we want to avoid. If that were the case, the hierarchical information wouldn't be used as much as it would be with a more balanced hierarchy.

For this implementation of a hierarchy, we have chosen not to assign the target labels to higher-level nodes and have the higher-level nodes be abstract instead. This way, only the leaves are the desirable classes or tags. It does however still allow us to use the hierarchical information. We end up with the 4 different clusters (COM-AC, COM-BK, SEDK-BK, and SEDK-AC) each of which have a total of 5 layers. First the root node, then a layer of 20 high-level abstract classes, then 60 mid-level abstract classes, next 130 low-level abstract classes, and finally 220 concrete classes, connected to the actual tags.

5.2 Classifiers

The classifiers were used in their described form as can be found in the Background section. Some changes did have to be made to the classifier code that could be found on GitHub, such that it could work with the rest of the code, our hierarchy and the dataset. Most of them were set up to work with a standard HMC testing dataset, which is often genetic data. A modification to HMC-LMLP was considered where the input vector was appended to each layer of the hierarchy. It was decided to not continue experiments with this variant as it would be impractical to train and testing showed that it didn't provide much improvement over the standard variant.

6 Experiment Design

With the experiment, we want to answer three research questions. These are answered using the previously described recommenders and hierarchies. These are trained with the dataset. When the training is done, predictions for a seperated subset of the dataset are made to compare to the actual values. These predictions are then evaluation with some metrics, which we can compare to the baseline. In the following subsections, the research questions, datasets, evaluation metrics and baseline are described.

6.1 Research Questions

With the experiment we want to answer the following questions:

RQ1: Can a HMC perform better compared to the baseline?

RQ2: What has more influence on the performance of a HMC, the recommender or the hierarchy?

RQ3: What impact does the construction of the hierarchy have on the performance of the model?

6.2 Dataset

The dataset used is a set of 152K GitHub repositories, each of which has a number of tags from the 220 derived from the preprocessing step developed by Izadi et al. [2]. The input data for the models consists of the repositories project names, descriptions, README files, wiki pages if present, and the file names concatenated together. Using each of these five data fields does increase the performance of the model: "adding more sources of information such as descriptions and file names indeed helps boost the models' performance" [2] (p. 25).

6.3 Evaluation Metrics

To evaluate the models, 20% of the dataset is set aside to compare performances. The input data for these datasets is fed through the model and the results are expressed for each repository as a chance for each tag to be relevant to that repository. For evaluating classifiers, precision and recall are standard options, which are calculated as follows:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

with TP, FP, and FN being the number of true positives, false positives, and false negatives, respectively.

However, for multilabel classification these metrics are insufficient. The output of the model is a chance between 0 and 1, so we need to set a threshold. Preferably, we use a metric that is independent of the quality of the threshold, as that is not what is being experimented with here. Instead, we can use precision and recall at k. These are defined as follows and calculated for every prediction:

$$P@k = \frac{|\{t_i \; with \; p_i \; among \; k \; highest \; predictions\}|}{k}$$

$$R@k = \frac{|\{t_i \; with \; p_i \; among \; k \; highest \; predictions\}|}{\min(k, \sum_i t_i)}$$

with $t_i = 0$ if label i is not assigned, $t_i = 1$ if label i is assigned and p_i the chance of label i being predicted by the algorithm. From this precision and recall, we can also calculate the F@k-value as the harmonic mean of the two. The F-value can be used to directly compare two pairs of precision and recall values.

Another method of evaluating the performance of multilabel classifiers is using the Average area Under Precision-Recall Curve (AUPRC). It measures the area under the precision-recall curve. This curve is obtained by first setting the threshold for predictions to 1. In this case, the recall is 0 and the precision is 1, as every prediction is negative. Then, the threshold is slowly lowered to 0, where the recall becomes 1 and the precision is 0, as every prediction is positive. The curve in the precision-recall graph obtained from this procedure is the Precision-Recall Curve. Next, we calculate the average area under it, which is the AUPRC [14]. Together, all these metrics should provide an accurate idea of how the performance of the algorithms compares to each other.

6.4 Baseline

As the goal here is to evaluate GitHub tag recommenders, a good but non-hierarchical recommender is needed as a baseline. From Izadi et al.'s 2021 paper we can see that using logistic regression in combination with a TF-IDF vectorizer gives the best results [2] in all the metrics. This makes it a simple and logical choice as a baseline for our experiments.

6.5 Implementation Details

For each stage of the process, the necessary details and parameters are provided in this section.

Hierarchy Generation

As discussed previously, two hierarchy generation methods were used: Agglomerative Clustering and Bisecting K-Means. Both AC and BK use the sklearn library [13]. AC is used with 'precomputed' affinity and 'average' linkage. BK uses a 'k-means++' for the initialization algorithm, targets the largest cluster when deciding which to bisect, and runs 16 times, each with different starting points, after which it chooses the best cluster.

Recommender Parameters

There are both general and specific parameters for the recommenders, which are detailed below.

General The input feature vector is generated with 24000 inputs, 20000 for training, and 4000 for testing. It is made with a Tf-Idf vectorizer from sklearn [13] and has 10000 features. Each recommender ran for 128 epochs, or less if there were time constraints. All recommenders were trained on the TU Delft "DelftBlue" high-performance cluster, which uses Intel XEON E5-6248R 24C 3.0GHz for the CPU nodes and NVIDIA Tesla V100S 32GB for the GPU nodes [15].

AWX AWX is built using Keras and uses a neural network. To train AWX we chose to have a network with an input vector of 10000 features, a hidden layer with 5000 features, and an output layer of 471 features, which corresponds to the tree.

Hierarchy	AWX	HMCN-F
Thin High Cluster	0,500	0,542
Thick High Cluster	0,498	0,526
Best hierarchy	0,546	0,570

Table 3: AUPRC scores for the simple hierarchies.

The first two layers use a hypertan activation function and the output layer uses a sigmoid activation. The "Adam" optimizer is used for optimizing and the loss function is binary cross-entropy.

C-HMCNN(h) C-HMCNN(h) is built with torch instead of Keras but is also a neural network. The input layer has 10000 features, the hidden layer 5000, and the output layer 471. It uses the "Adam" optimizer with a learning rate of 10^{-4} and weight decay of 10^{-5}. For the activation function it uses the rectified linear unit function (ReLU): $f(x) = max(0, x)$.

HMC-LMLP Details for the number of layers in HMC-LMLP can be found in the implementation section. For the neural networks, the "MLPClassifier" from sklearn [13] was used, also with the "adam" solver. The default activation function and learning rate were used, which are ReLU and 10^-3 respectively.

HMCN-F For details on what size certain neural network layers are, the original paper by Wehrmann et al. provides all the necessary details [6]. Again, the "Adam" optimizer is used, with a learning rate of $2 * 10^-4$. The default value for $\beta = 0,5$, which controls the balance between global and local influence, was kept.

LR The sklearn library also has an implementation for logistic regression, using the "MultiOutputClassifier" with logistic regression [13]. The only parameter that was changed is the class weight, which was changed to balanced.

7 Results

To answer **RQ1**, we can compare the AUPRC scores of the combinations of recommenders and hierarchies with that when using logistic regression. These scores are aggregated in Table 4. The highest AUPRC score for the HMCs is 0,570 when using HMCN-F with SEDK-AC. The score for LR is 0,556, which gives a difference of 0,014. It is a small difference, but it does show that HMCs can surpass LR in terms of AUPRC.

To answer **RQ2**, we can also again look at the same table. We see that for these recommenders and hierarchies, the recommenders have a larger influence on the AUPRC. To put this more formally: the main cause of deviation between AUPRC scores seems to be the change in recommender and not the hierarchy.

However, these hierarchies are all structured very similarly. They all consist of 5 layers including the root and leaves and have 20, 60 and 130 nodes in the layers between the root and leaves. There are other ways to construct these hierachies, with different amount of nodes, or even using less or no abstract nodes. These fall outside of the scope of this research however, so for this context we can conclude that the larger influence on the performance is the recommender.

Finally, we answer **RQ3**. From Table 4 we can't conclude that one of the hierarchies always gives significantly better results overall. The Co-Occurence matrix seems to perform worse for C-HMCNN(h), but does quite well with HMC-LMLP, at least when combined using Agglomerative Clustering. With Bisecting K-Means it performs the worst for that recommender. From these results we can't make claims on the impact of the hierarchy on the performance.

What we can do however, is compare these hierarchies to trivial hierarchies. Two additional hierarchies were constructed. The first is a hierarchy where instead of using 20, 60 and 130 as layer sizes, we use only 1s. The second uses a similar idea, but has 220 nodes at every layer. We compare the AUPRC scores of these three hierarchies to that of the best score with AWX and HMCN-F. These results are presented in Table 3. We see that there is some improvement in using a non-trivial hierarchy. AWX gives an increase of 0,046 when using a hierarchy and HMCN-F 0,028. It seems that using a non-trivial hierarchy does yield some improvement in performance over using a non-trivial one, but in this case it is of small significance.

7.1 Sample Recommendations

In order to provide additional insight into what the results from these models look like a sample is presented. In Table 5 there are a total of 10 predictions from logistic regression and HMCN-F with SKG-AC, as well as the true labels. As the predictions from the models are probabilities, we decided to take the top 3 predictions for each recommender. We take two rows as examples: the 2nd and the 8th.

With the 2nd row, there is a significant amount of mismatch between the labels. Express, JWT and KOA all are absent in the predictions, with HMCN-F not retrieving any of the true labels. Conceptually however, a project with JWT (JSON web tokens), KOA (a web framework) and security can reasonably be expected to be written in Javascript and use HTTP. Even if the true labels weren't predicted, the predicted labels would fit with the project.

With the 8th row, Bootstrap and HTML are present in the true labels, but not in the predictions. In fact, both LR and HMCN-F predicted Composer instead. One might expect Composer to be close to Bootstrap and HTML in the hierarchy. However, this is not the case as Composer does not share a high-level cluster with Bootstrap and HTML.

In this research we didn't perform a qualitative analysis, instead focussing more on a statistical one. However, a qualitative analysis might show that while some recommenders are not very good at retrieving the exact same set of labels as the true labels, they do provide labels that could fit with a repository.

8 Discussion

The results are not as conclusive as we might like. The differences between the baseline and best-performing HMC are very small. For precision and recall at k with k being 3 or 5 the baseline even outperforms the best performing HMC, as

Hierarchy	Classifier				
	AWX	C-HMCNN(h)	HMC-LMLP	HMCN-F	LR
COM-AC	0,542	0,357	0,128	0,566	0,556
COM-BK	0,546	0,355	0,091	0,568	-
SEDK-AC	0,542	0,372	0,107	**0,570**	-
SEDK-BK	0,539	0,373	0,121	0,564	-

Table 4: AUPRC scores for all hierarchy-recommender combinations, with LR placed at COM-AC for convenience

True labels	**LR**	**HMCN-F**
monitoring	API, Ethereum, monitoring	cryptocurrency, Ethereum, monitoring
express, JWT, KOA, security	HTTP, Javascript, security	HTTP, Javascript, node.js
Java	framework, Java, library	framework, Java, Python
Android, Firebase, Flutter, iOS, mobile	Dart, Firebase, Flutter	Dart, firebase, Flutter
algorithm, api, bot, Python	algorithm, cryptocurrency, Python	Bitcoin, Javascript, Python
Docker, server	Docker, server, shell	Docker, server, shell
Dart, Flutter	Dart, Flutter, library	Dart, Flutter, library
Bootstrap, HTML, Laravel, PHP	Composer, PHP, Laravel	Composer, PHP, Laravel
Scala	library, Java, Scala	library, machine-learning, Scala
AI	AI, machine-learning, Python	AI, deep-learning, machine-learning

Table 5: Comparison of 10 tuples of true labels, LR predictions and HMCN-F predictions, using the top 3 results for the recommenders

can be seen in Table 6. What we did learn is that HMCN-F is a good HMC to be used for repository recommendation. With the limited experiments done in this research, it managed to perform nearly as well as logistic regression, which, one could argue, is a rather high baseline. We did choose to continue using LR as a baseline as the goal of the research was to find out if HMCs can outperform non-hierarchical classifiers, of which LR is the best performing [2].

The other HMCs also provide interesting results. AWX comes very close to the baseline, but cannot outperform it. It does seem to have a larger impact on the quality of the hierarchy compared to HMCN-F, so perhaps with a better hierarchy, it can outperform the baseline. C-HMCNN(h) and HMC-LMLP fall far behind the other two. For HMC-LMLP we can think of a reason why this happens. In the first layer of the model, you can say that all the 10000 features get ”compressed” down into just 20. From those 20 features, the rest of the features must be inferred. This amount is just too limited to yield accurate results in the lower hierarchies. This problem is fixed in the later iterations such as HMCN-F, where the input at each layer is not just the previous layer, but also the input feature vector

9 Responsible Research

This section is split in three parts: data collection, code usage and reproducibility. For all of these parts, we will discuss what the ethical risks are and how these are mitigated.

9.1 Data Collection

To train the recommender systems and build the hierarchies, data has to be used. All the data was retrieved from GitHub’s public API. As the same dataset was used as by Izadi et al. in 2021, their paper has a more detailed description of how the raw information from the API was used to create the final dataset. As this is all public, non-personal, information it is deemed that there are no ethical issues with using it. However, the dataset itself was not investigated and checked for any unethical items, whatever that may entail. It is not impossible that data was used from repositories that could be considered unethical, but the effect of such repositories on this research is absolutely minimal and practically zero.

9.2 Code Usage

In order to train the different models, code was used from other people. Specifically, for AWX, HMCN-F and C-HMCNN(h), repositories were found on GitHub (not using tags) with the code for these models present, which was then used. LR and HMC-LMLP were both built using the sklearn Python library.

9.3 Reproducibility

The experiment effectively comes down to running a Python program. As this is available at https://github.com/AvdRande/AvdR-TUD-RP, it should be doable for anyone with spare computing power to train and run the models. Running the experiment with new data is probably more difficult, as that would require a new dataset from GitHub and preprocessing as described by Izadi et al. [2]. With such a dataset, it should be possible for a programmer with a third year computer science student level of skill to run the experiment.

10 Future Work

The current experiment was limited in quite some dimensions, such as the recommenders, hierarchies, and the dataset. Each of these could be expanded and investigated further. Other HMCs might be more fitting for a task such as this one (maybe examples?). To us, it seems that most improvement can come from other ways of building a hierarchy. For this paper, hierarchies are used where only the leaves correspond to actual tags. While it may not be reasonable to assign actual tags to all abstract nodes, it would probably work better with the strengths of HMCs. Alternatively, one could also experiment with using a DAG instead of a tree, as HMCs such as AWX and C-HMCNN(h) can already work with these as well. They might better represent the structure of tags in GitHub. For example, one might consider $javascript$ to be both a child of $programming\text{-}language$ and $web\text{-}development$. Finally, the used dataset has 220 tags for every repository, which is significantly less than the number of tags used in the platform or even the number of curated tags, which is 709 at the time of writing [16]. Using more tags and more data points will always yield more accurate results and in this case, might allow for a better hierarchy construction.

11 Conclusion

The goal of this paper was to find out if HMCs can be used to improve over non-hierarchical classifiers when assigning tags to GitHub repositories. We also wanted to find out which HMCs and hierarchies work the best and which of those two has the bigger influence on performance.

From the experiments we conclude that using HMCN-F yields the best results of the HMCs, but it barely outperforms the baseline, logistic regression, using AUPCR as a performance measurement. AWX also produces relatively good results but is slightly worse compared to the baseline. Between recommenders and hierarchies, the recommenders seem to have a larger impact on the performance of the model. The construction of the hierarchy using our approach has such a low impact on the performance that even using a trivial or simple hierarchy with HMCN-F has AUPCR scores comparable to the baseline. To be clear: with HMCN-F the AUPCR scores when using a constructed hierarchy versus a simple hierarchy differ by only 0,028.

References

[1] Justine Lyon, *Market Share of Github*, Jun. 2022. [Online]. Available: https://www.slintel.com/tech/source-code-management/github-market-share.

[2] M. Izadi, A. Heydarnoori, and G. Gousios, "Topic recommendation for software repositories using multi-label classification algorithms," *Empirical Software Engineering*, vol. 26, no. 5, p. 93, Sep. 2021, ISSN: 1382-3256. DOI: 10.1007/s10664-021-09976-2. [Online]. Available: https://link.springer.com/10.1007/s10664-021-09976-2.

[3] Y. Zhang, F. F. Xu, S. Li, *et al.*, "HiGitClass: Keyword-Driven Hierarchical Classification of GitHub Repositories," in *2019 IEEE International Conference on Data Mining (ICDM)*, vol. abs/1910.07115, IEEE, Nov. 2019, pp. 876–885, ISBN: 978-1-7281-4604-1. DOI: 10.1109/ICDM.2019.00098. [Online]. Available: https://ieeexplore.ieee.org/document/8970799/.

[4] L. Masera and E. Blanzieri, "AWX: An Integrated Approach to Hierarchical-Multilabel Classification," Tech. Rep., 2018, pp. 322–336.

[5] E. Giunchiglia and T. Lukasiewicz, "Coherent Hierarchical Multi-Label Classification Networks," Oct. 2020. [Online]. Available: http://arxiv.org/abs/2010.10151.

[6] J. Wehrmann, R. Cerri, and R. C. Barros, "Hierarchical Multi-Label Classification Networks," Tech. Rep., 2018. [Online]. Available: https://proceedings.mlr.press/v80/wehrmann18a.html.

[7] M. Izadi, M. Nejati, A. Heydarnoori, M. Izadi, and A. Heydarnoori, "Semantically-enhanced Topic Recommendation Systems for Software Projects," Tech. Rep., 2022.

[8] D. D. Lewis, Y. Yang, T. G. Rose, F. Li, and F. Li LEWIS, "RCV1: A New Benchmark Collection for Text Categorization Research," *Journal of Machine Learning Research*, vol. 5, pp. 361–397, 2004, ISSN: 1532-4435.

[9] L. Schietgat, C. Vens, J. Struyf, H. Blockeel, D. Kocev, and S. Džeroski, "Predicting gene function using hierarchical multi-label decision tree ensembles," *BMC Bioinformatics*, vol. 11, no. 1, p. 2, 2010, ISSN: 1471-2105. DOI: 10.1186/1471-2105-11-2. [Online]. Available: https://doi.org/10.1186/1471-2105-11-2.

[10] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996, ISSN: 1573-0565. DOI: 10.1007/BF00058655. [Online]. Available: https://doi.org/10.1007/BF00058655.

[11] W. Bi and J. T. Kwok, "Multi-Label Classification on Tree- and DAG-Structured Hierarchies," Tech. Rep., 2011, pp. 17–24.

[12] R. Cerri, R. C. Barros, and A. C. De Carvalho, "Hierarchical multi-label classification using local neural networks," in *Journal of Computer and System Sciences*, vol. 80, Academic Press Inc., Feb. 2014, pp. 39–56. DOI: 10.1016/j.jcss.2013.03.007.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine Learning in {P}ython," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] Rachel Draelos, *Measuring Performance: AUPRC and Average Precision*, Mar. 2019. [Online]. Available: https://glassboxmedicine.com/2019/03/02/measuring-performance-auprc/.

[15] D. H. P. C. C. (DHPC), *DelftBlue Supercomputer (Phase 1)*, https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

[16] GitHub, *GitHub Explore Topics*, Jun. 2022. [Online]. Available: https://github.com/github/explore/tree/main/topics.

A Complete results

In the results chapter only the AUPRC scores are presented, here are also the precision and recall scores, both with top 1, 3 and 5. Additionally, the F scores are also provided, which is the harmonic mean between precision and recall.

Hierarchy	**Recommender**	**P@1**	**P@3**	**P@5**	**R@1**	**R@3**	**R@5**	**AUPRC**	**F@1**	**F@3**	**F@5**
COM-AC	AWX	71,1%	42,7%	29,4%	71,1%	73,1%	78,6%	0,542	71,1%	53,9%	42,8%
COM-BK	AWX	71,4%	42,4%	29,2%	71,4%	72,9%	78,1%	0,546	71,4%	53,6%	42,5%
SKG-AC	AWX	71,0%	42,2%	29,0%	71,0%	72,4%	77,4%	0,542	71,0%	53,3%	42,2%
SKG-BK	AWX	70,9%	42,2%	29,0%	70,9%	72,6%	77,9%	0,539	70,9%	53,3%	42,2%
thick_high	AWX	73,0%	44,6%	29,2%	73,0%	75,9%	83,4%	0,498	73,0%	56,2%	43,2%
thin_high	AWX	73,3%	44,6%	31,3%	73,3%	76,0%	83,0%	0,500	73,3%	56,2%	45,4%
COM-AC	C-HMCNN(h)	65,6%	38,6%	27,5%	65,6%	66,0%	73,4%	0,357	65,6%	48,7%	40,0%
COM-BK	C-HMCNN(h)	64,2%	38,6%	27,4%	64,2%	65,9%	72,8%	0,355	64,2%	48,7%	39,8%
SKG-AC	C-HMCNN(h)	66,5%	40,1%	28,4%	66,5%	67,8%	75,0%	0,372	66,5%	50,4%	41,2%
SKG-BK	C-HMCNN(h)	66,0%	39,7%	27,9%	66,0%	67,2%	74,2%	0,373	66,0%	49,9%	40,6%
COM-AC	HMC-LMLP	26,2%	19,2%	15,4%	26,2%	32,8%	41,8%	0,128	26,2%	24,3%	22,5%
COM-BK	HMC-LMLP	21,2%	16,2%	12,9%	21,2%	28,4%	35,5%	0,091	21,2%	20,7%	18,9%
SKG-AC	HMC-LMLP	29,7%	20,2%	15,4%	28,7%	34,4%	42,0%	0,107	29,2%	25,4%	22,6%
SKG-BK	HMC-LMLP	30,1%	20,8%	15,7%	30,1%	35,6%	43,0%	0,121	30,1%	26,2%	23,1%
COM-AC	HMCN-F	74,4%	43,5%	29,9%	74,4%	74,4%	79,8%	0,566	74,4%	54,9%	43,5%
COM-BK	HMCN-F	74,1%	43,3%	29,9%	74,1%	74,2%	80,0%	0,568	74,1%	54,7%	43,5%
SKG-AC	HMCN-F	73,7%	43,2%	29,7%	73,7%	73,9%	79,7%	**0,570**	73,7%	54,5%	43,3%
SKG-BK	HMCN-F	**74,7%**	43,3%	29,8%	**74,7%**	74,3%	79,9%	0,564	**74,7%**	54,7%	43,4%
thick_high	HMCN-F	72,3%	41,6%	28,4%	72,3%	71,6%	76,7%	0,526	72,3%	52,6%	41,4%
thin_high	HMCN-F	72,4%	42,6%	29,2%	72,4%	73,1%	78,4%	0,542	72,4%	53,8%	42,5%
	LR	73,5%	**45,5%**	**31,7%**	73,5%	**77,3%**	**83,9%**	0,556	73,5%	**57,3%**	**46,0%**

Table 6: Complete results of the experiment, where thick_high and thin_high are two simple hierarchies.