Automated Tagging for the Retrieval of Software Resources in Grid and Cloud Infrastructures

Ioannis Katakis
University of Cyprus
katak@cs.ucy.ac.cy

George Pallis
University of Cyprus
gpallis@cs.ucy.ac.cy

Marios D. Dikaiakos
University of Cyprus
mdd@cs.ucy.ac.cy

Onisiforos Onoufriou
University of Cyprus
cs07oo2@cs.ucy.ac.cy

Abstract

A key challenge for Grid and Cloud infrastructures is to make their services easily accessible and attractive to end-users. In this paper we introduce tagging capabilities to the Minersoft system, a powerful tool for software search and discovery in order to help end-users locate application software suitable to their needs. Minersoft is now able to predict and automatically assign tags to software resources it indexes. In order to achieve this, we model the problem of tag prediction as a multi-label classification problem. Using data extracted from production-quality Grid and Cloud computing infrastructures, we evaluate an important number of multi-label classifiers and discuss which one and with what settings is the most appropriate for use in the particular problem.

1. Introduction

A growing number of large-scale Grid and Cloud infrastructures are in operation around the world, providing production-quality computing and storage services to numerous users from a wide range of scientific and business fields. With the rapid proliferation of Cloud and Grid application services, it is important to make these services easily accessible and attractive to end-users. To achieve this goal, advanced, user-friendly tools for software search and discovery should be established [1]. To motivate the importance of software searching tools, let us consider a researcher searching for graph mining software deployed on a Grid/Cloud infrastructure. Unfortunately, the manual discovery of such software is a daunting, nearly impossible task. Taking the case of EGEE/EGI, one of the largest production Grids currently in operation, the researcher would have to gain access and search inside 300 sites, several of which host well over 1 million software-related files. The situation is equally challenging in emerging Cloud infrastructures: the Amazon Elastic Cloud provides access to a growing repository with more than 2,000 virtual computational servers (AMIs), with each AMI comprising over 14,000 files, including application and utility software. Therefore, the researcher would have to spawn some AMIs, connect to them, and search manually for installed software.

Following this *motivation* and taking account that software installed in Grid/Cloud infrastructures is unstructured and software-related metadata or software descriptions in natural language are typically poor or unavailable, we developed the *Minersoft* software search engine. A prototype implementation of Minersoft is available at http://euclid.grid.ucy.ac.cy:1997/MinerSoft/SimpSearch. Unlike desktop search, which is designed to assist users in locating specific files, Minersoft supports searching not only for source code but also for executables and libraries stored in binary format, and metadata situated in Cloud and Grid Infrastructures. Minersoft visits multiple computational resources, crawls the file systems of Grid and Cloud computing sites, identifies software files of interest (binaries, libraries, documentations etc), assigns type information to these files, and discovers implicit associations between them. Subsequently, it extracts words from the context that surrounds software files, in order to annotate the files with descriptions amenable to full-text search. This work continues and improves upon the authors' preliminary efforts in [2], [1].

Tagging thrives in Web sites with user-submitted content where tags are voluntarily assigned for information retrieval purposes. In most cases, users can do tag-based searches or browse objects of a particular tag. Tags are currently assigned to many different types of information sources such as, images (Flickr), videos (YouTube) and music (Last.fm). Recently, tagging has made an appearance in software repositories like Google Code.

We exploit tagging and automated tagging in or-

der to improve software retrieval in Cloud and Grid infrastructures. User-entered tags can add valuable searchable meta-data to software files. For example, a user can assign tags that represent general concepts like "physics" and "data mining" that might not be able to be extracted by Minersoft if those terms are not appearing in the files (or in associated files). By extending Minersoft in this direction, we allow the user to assign tags to resources and, most importantly, the user is now able to submit tag-based queries.

However, there are two important issues concerning tagging: a) users are not always willing to submit tags and the number of tags that they enter is usually small; and b) users may select different words for expressing the same concept or the same word for different concepts. This creates a noisy tag space and makes it harder to find software tagged by other users.

A key question is "how can we improve the tagging process in Minersoft in order to avoid the aforementioned obstacles?" To address it, we introduced a machine learning approach that is able to "learn" from user-tag examples and automatically assign tags to new software resources. For instance, consider a user who wants to search for "sympy" - a multimedia library for Python. Minesoft returns a description of this file with a listing of computational resources where this software has been located. Using the proposed auto-tagging approach, numerous tags will be automatically assigned to this software (e.g., python, symbolic, calculation, algebra, differentiation). The value of tagging process is that searching software resources in Clouds/Grids is significantly improving since some of these tags are not included in the description of this file. The *main contributions* of this work can be summarized as follows:

- We extend Minersoft by enabling users to a) enter their own tags into software files installed in CloudGrid infrastructures and b) execute tag-based queries - in addition to keyword-based queries.
- We introduce a machine learning approach for automatically assigning tags to software resources that have no user-entered tags. Specifically, we model the problem of tag recommendation as a multi-label classification task. The advantages of this approach can be summarized as follows: a) it is content-based and therefore there is no cold-start problem that limits the user-based and item-based approaches (tag recommendations can be produced even for new files), and, b) the method is generic and any multi-label classifiers can be used, based on application requirements and available resources.
- We conduct an experimental evaluation of Minersoft's automated tagging on real, large-scale Grid/Cloud testbeds, exploring performance issues of the proposed approach. We evaluate five multi-label classifiers on datasets of software files installed in Grid/Cloud infrastructures with varying parameter settings. Through the evaluation, we provide guidelines for selection of the most appropriate classifier.

The structure of the paper is as follows. Section 2 presents an overview of related work. In Section 3, we formulate the problem of automated tagging as a multi-label classification task. Sections 4 and 5 describe the experimental setup and the evaluation results respectively. Section 6 presents the implemented automated tagging features in Minersoft whereas Section 7 presents comments on the utility of tagging.

2. Related Work

This section provides an overview of the related work in the area of software retrieval in Grid and Cloud infrastructures. We also present cases where tagging is used in software resources and some representative approaches in automated tagging.

2.1. Software Retrieval in Grids and Clouds

A number of research efforts [3], [4] have investigated the problem of software-component retrieval in the context of language-specific software repositories and CASE tools. Nowadays, software retrieval in Clouds has become an important element in software development, where software components and libraries are used extensively in order to harness the capabilities of these infrastructures. To the best of our knowledge, Minersoft [1], [2] provides the first full-text search facility for the retrieval of software installed in large-scale Grid and Cloud infrastructures. Minersoft differs from prior works on software retrieval [5], [3], [6] that use the keyword paradigm in a number of key aspects. Specifically, Minersoft supports searching for software installed in file systems of distributed infrastructures (Grids, Clouds, clusters), as well as in software repositories. Minersoft supports searching not only for source code but also for executables and libraries stored in binary format, and metadata (software versions, timestamps, permissions). In addition, Minersoft addresses a number of additional implementation challenges that are particular to Grid and Cloud infrastructures. Specifically, harvesting is distributed to the computational resources available in the infrastructure, achieving load balancing and reducing data communication overhead

between the search engine and Grid or Cloud sites. Finally, Minersoft's architecture and implementation adopts a non-intrusive approach, which facilitates the deployment of the system on different Grids and Clouds.

2.2. Tagging Software and Automated Tagging

An analysis of the tagging behavior of developers and the tagging features of development environments are presented in [7]. The focus of these research studies is on how tagging can aid collaborative development tools. TagSEA [8] is a tool that supports developers in annotating source code and can be beneficial in software development and maintenance. IBM Jazz[1] allows developers to enter keywords in order to annotate source code. Contrary to our solution, these approaches focus on tagging code segments and do not provide any automated tagging capability.

Many research efforts have been made towards the direction of automatically recommending tags. In [9] tag recommendations are provided for the Flickr image sharing social network based on the co-occurrence of the tags. A similar approach is proposed in [10] where tags are assigned to web-services in order to improve web service discovery. However in the last two approaches in order to recommend tags the user has to enter some tags first. In [11], a content-similarity based method for tag-recommendation for the del.icio.us web site (a bookmark sharing social network) has been proposed. This approach, being content-based is not limited by the cold-start problem, however, it can not be generalized using other machine learning techniques.

3. Machine Learning Automated Tagging

In order to develop the automated tagging features of Minersoft, we followed a machine learning approach. More specifically we modeled the problem as a multi-label classification task. The general architecture of the tagging-enhanced Minersoft can be seen in Figure 1 (dotted lines represent the new automated tagging components and procedures added to MinerSoft).

3.1. Multi-Label Classification Background

Supervised learning is the machine learning task of inferring a function from a set of *examples* (training data). Each example is a tuple consisting of an input object $\overrightarrow{x}$, and an output value which is called *label* or *class*. Each object $\overrightarrow{x}$ is characterized by a set of n attributes $\overrightarrow{x}$=$(x_1, x_2, \ldots, x_n)$. A supervised learning algorithm analyzes the training data and infers a function called *classifier*. The classifier can predict the correct output value for any input object.

A large body of research in supervised learning deals with the analysis of *single-label* data, where training examples are associated with a single label from a set of labels L. However, training examples in several application domains (like tagging) are often associated with a *set* of labels $Y \subseteq L$. Such data are called *multi-label*. Textual data, such as documents and web pages, are frequently annotated with more than a single label.

Recently, the issue of learning from multi-label data has attracted attention from a lot of researchers, motivated from an increasing number of new applications, such as semantic annotation of images and video [12]. Multi-label learning methods can be grouped into three categories [12]: i) *problem transformation*, ii) *algorithm adaptation*, iii) *ensemble methods*. Methods of the first group transform the learning task into one or more single-label classification tasks, for which a large bibliography of learning algorithms exists. The second group of methods extends learning algorithms in order to handle multi-label data directly. Finally, in the third group, approaches are combined in order to achieve higher predictive accuracy.

3.2. Problem Modeling

In order to model the problem of automated tag annotation of software resources we need to transform these resources into attribute vectors (see Section 3.1). To achieve this, Minersoft invokes file-system utilities and object-code analyzers, implements heuristics for file-type identification and filename normalization, and performs document analysis algorithms on software documentation files and source-code comments. The results of Minersoft harvesting are encoded in the Software Graph (a weighted, typed graph that represents software files and their associations in a single data structure), which is used to represent the context of discovered software files. We process the Software Graph to annotate software files with metadata and keywords, and use these to build an inverted index of software. Indexes from different computational resource providers in Grids and Clouds are retrieved and merged into a central inverted index, which is used to support full-text software retrieval. The inverted index consists of the software resources, where each software resource s_i is represented as an attribute vector $\overrightarrow{s_i} = \{w_{(i,1)}, \ldots, w_{(i,|V|)}\}$, where $w_{(i,j)}$ is the weight of the word j for the file i, and $|V|$ is the

1. http://jazz.net

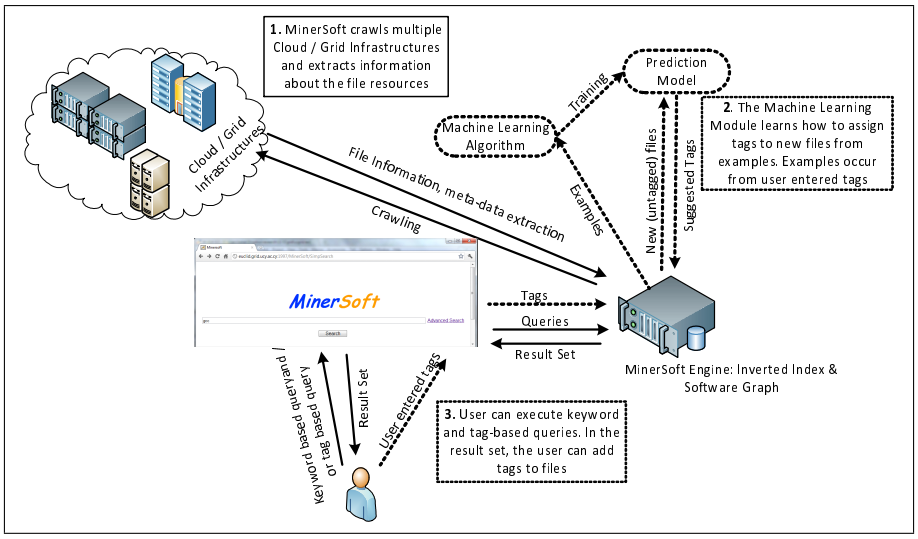

Figure 1. MinerSoft and Tagging

size of the vocabulary (the set of all distinct words). The value of $w_{(i,j)}$ is based on the well-known tf-idf measure. Figure 2 depicts the procedure of converting the software files existing in Clouds/Grids into training examples suitable for the machine learning algorithm.

Labels for the classification problem are the tags that the users of Minersoft assign to each software file. In order to create training data, we have extracted the keywords and tags that have been used by the EGI administrators in order to describe the installed applications on ATLAS Virtual Organization (VO) of EGI infrastructure (http://appdb.egi.eu/).

4. Experimental Evaluation

The purpose of the evaluation is to a) estimate the predictive performance that can be achieved in the automated tagging process using multi-label classification methods, and b) evaluate an important number of multi-label classifiers in order to find out the most appropriate for this specific problem. For the above reasons, we have performed evaluation of five well known classifiers and study them in terms of quality of prediction and time performance.

Quality of prediction is estimated using the *micro-F measure*. The F-measure is a widely used metric for information retrieval and classification tasks since it captures precision and recall and it is not sensitive to label imbalance. Given the number of true positives (tp), true negatives (tn), false positives (fp) and false negatives (fn), F is defined as follows:

$$F = \frac{2 * tp}{2 * tp + fp + fn} \quad (1)$$

With micro-averaging, binary (two-label) evaluation measures can be calculated across several labels. Consider a binary evaluation measure $B(tp, tn, fp, fn)$. Let tp_λ, fp_λ, tn_λ and fn_λ be the number of true positives, false positives, true negatives and false negatives after binary evaluation for a label λ. The micro-averaged version of B, is calculated as follows:

$$B_{\text{micro}} = B\left(\sum_{\lambda=1}^{M} tp_\lambda, \sum_{\lambda=1}^{M} fp_\lambda, \sum_{\lambda=1}^{M} tn_\lambda, \sum_{\lambda=1}^{M} fn_\lambda\right) \quad (2)$$

Time Performance is measured in CPU time (seconds). It is the time necessary for a method to learn from a set of examples and classify the unknown cases.

We consider both of the above factors equally important for selecting the classifier for creating the automated tagger of Minersoft.

4.1. Testbed Description

We have deployed and operated Minersoft on real production-rate testbeds. In particular, our testbed includes software installed on a) ATLAS of EGI infrastructure; b) 20 Virtual Servers of the Amazon Elastic Computing Cloud; c) 10 Virtual Servers of the Rackspace Cloud. To evaluate the effectiveness of automated tagging feature of Minersoft, we use the files installed on ATLAS. Note that EGI administrators have included short descriptions for the software that has been installed on ATLAS. We consider the words in these descriptions as tags in order to create the examples for the machine learning algorithms. After the evaluation we apply the classifiers learned from

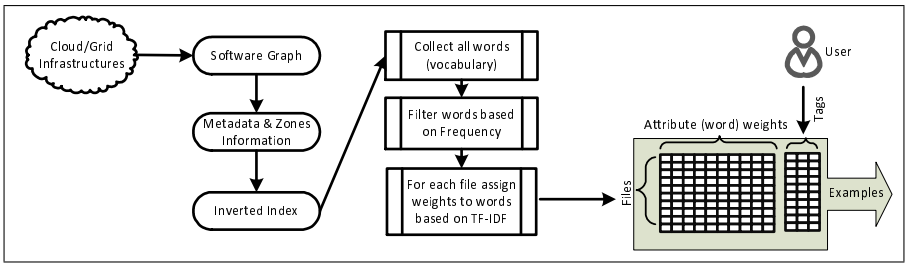

Figure 2. Attribute Extraction and Examples Construction

Dataset	Attr	Tags	AvgT	TCO
D_1	500	250	16.6	5760
D_2	1000	250	16.5	5803
D_3	1500	250	17.0	5718
D_4	2000	100	10.9	4964
D_5	2000	150	13.4	5415
D_6	2000	200	14.3	5396
D_7	2000	250	16.8	5798
D_8	2000	300	17.8	5811
D_9	2000	350	19.2	5959
D_{10}	2000	400	19.5	5999
D_{11}	2500	250	16.9	5763
D_{12}	3000	250	16.8	5859
D_{13}	3500	250	16.8	5749

Table 1. Datasets used in the experiments (Attr: Number of attributes, Tags: Number of tags, AvgT: Average number of tags per instance, TCO: number of different tag combinations observed)

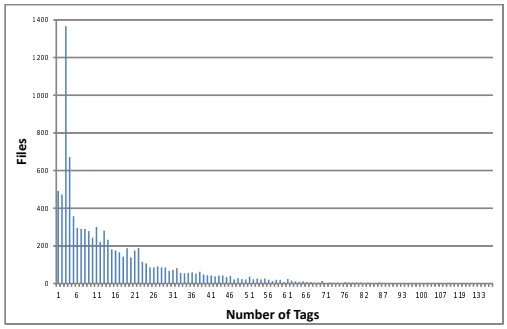

Figure 3. Number of labels distribution

ATLAS data to all software files that are indexed in Minersoft (see Section 6).

We split randomly the initial set of examples into *training* and *test* set (50% split). The classifier is built on the trained set and then evaluated by comparing its predictions with the ground truth labels of the testing set using micro-F. We then switch train with test and repeat the procedure. Finally we compute the average of the two obtained values of micro-F.

In order to study the scalability of each method we have created datasets with varying number of attributes (i.e. words) and varying number of labels that are selected to be included in the dataset (the selection in both cases was based on frequency of appearance). Both factors, number of labels and attributes, can significantly affect the performance in quality of prediction and time performance. We have created two series of datasets (13 datasets in total), the first one with varying number of labels (tags) (from 100 to 400 with a step of 50) and the second one with a varying number of attributes from 500 to 3500 with step of 500). In all cases the number of examples is 10,000 files. Table 1 presents information about all datasets.

Figure 3 depicts the distribution of the number of tags (labels) per file, in one of the datasets (200 labels, 1000 attributes). We observe that in terms of label distribution the dataset simulates quite sufficiently the long-tail distribution that appears in many tagging systems i.e. a great number of files are tagged with a small number of tags whereas only a small number of files are annotated with large number of tags.

4.2. Algorithms

In the evaluation, we included five multi-label classification methods. We have tried to include at least one representative from each category (see section 3.1).

- *BR* - Binary relevance is a popular problem transformation method that learns one binary classifier for each label [12]
- *LP* - Label powerset considers each unique set of labels that exists in a multi-label training set as one of the classes of a new single-label classification task [12]
- *MLkNN* - Multi-label kNN is based on the popular k Nearest Neighbor lazy learning algorithm [13]
- *BPMLL* - Back Propagation Multi-Label Learning is an adaptation of the popular back-propagation algorithm [14]
- *RAkEL* - Random k-labelsets, breaks the initial set of labels into a number of small random

subsets, called labelsets and employs LP to train a corresponding classifier [15]

For all methods, we used the implementations of the Mulan [16] software library.

5. Results

In this section we present and discuss the results of the experimental evaluation.

5.1. Varying the Number of Labels

Figure 4 displays the micro-F measure against the varying number of labels. The first observation concerns the superiority of BR and Rakel over all other methods. BR and Rakel have presented high predictive performance in many cases [15]. However the high value of F-measure in this particular case is due to the fact that many popular labels are appearing in many files. This characteristic of the data was exploited by BR and Rakel. BR presents high predictive performance because it divides the classification tasks into simpler one-against-all tasks. This feature is a critical advantage especially in cases when the correlation between labels is weak. Rakel on the other hand, manages to provide accurate decisions because of its ensemble nature. BPMLL presents particulary low prediction accuracy. As noted in other studies [15], BPMLL can present low performance in some cases due to its many parameters needing optimization. Concerning the scalability with the number of labels, we observe that all methods present a decrease in their prediction quality since more labels lead to a more complex problem.

In Figure 5 we observe the CPU time needed for training and classification for each method against the number of labels. In this case, we observe that LP is an undemanding method in terms of computational resources. This observation can be easily explained as LP only maintains a single label classifier. On the other hand, Rakel's nature as an ensemble of multi-label classifiers comes with a high computational cost. BPMLL is also a time consuming algorithm since it is based on a more complex Neural Network classifier. Concerning the variation of performance with the number of labels, we observe that MLkNN and LP provide better scalability and seem not to be affected by the increase in labels. This again can be explained by considering the good scalability of the single-label classifiers that MLkNN and LP are based (kNN and J48 respectively). The worst scalability in this case is presented by neural network based BPMLL. The dataset with 200 labels leads to an increase in CPU

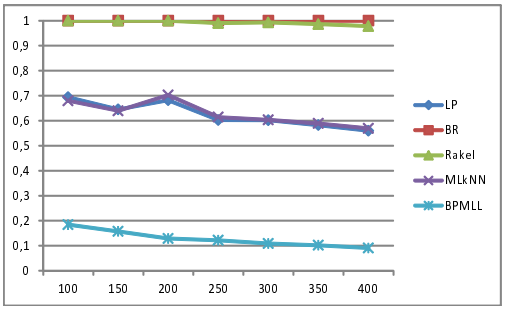

Figure 4. F-measure vs Number of Labels

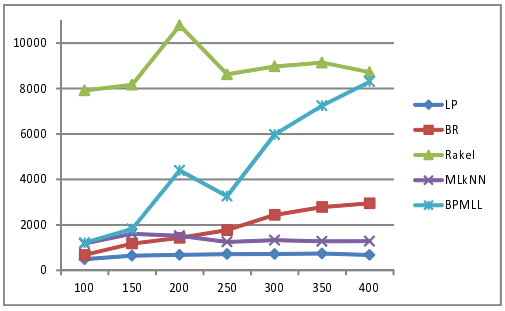

Figure 5. CPU Time vs Number of Labels

time in Rakel and BPMLL. This could be explained by the different label combinations in this case that lead to a more complex problem. This fact affects BPMLL and Rakel since they are sensitive to number of labels.

5.2. Varying the Number of Attributes

In this section we study the scalability of the methods in terms of the number of words that are used as input attributes. Figure 6 displays the F-measure of all methods against the varying number of attributes. The advantage of BR and Rakel in quality of prediction is verified from this figure as well. From this graph we observe, after an initial increase, no important variations for BR and Rakel. However, MLkNN and LP seem to have small decrease in predictive performance. A small increase is expected when the learning method is able to absorb the information in the extra attributes (like BR and Rakel). However extra attributes could lead to decrease in accuracy due to the higher complexity of the problem and noise that the method is not able to isolate (like MLkNN and LP).

By studying Figure 7 we could conclude that all methods present an increase in time as the number of attributes increases. This can be explained by the complexity of a problem with a high number of attributes.

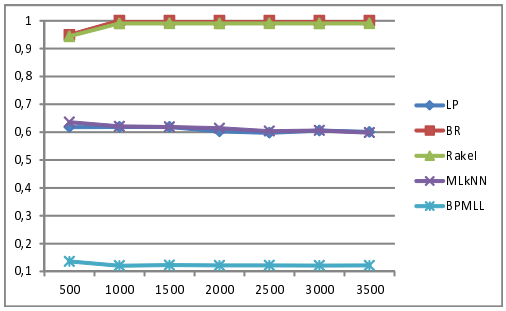

Figure 6. F-measure vs Number of Attributes

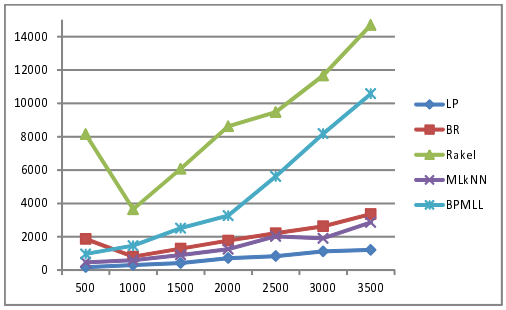

Figure 7. CPU Time vs Number of Attributes

6. Automated Tagging in Practice

Summarizing the important points of the evaluation, we should note that the most accurate methods are BR and Rakel with BR being the most efficient. In addition, BR has also good scalability in terms of attributes and labels. Based on these results, we select BR to be integrated in Minersoft in order to provide automated tagging functionality. Our goal is to enrich the tags that are assigned into software files and therefore enhance the retrieval performance of Minersoft.

In this context, we have used the BR algorithm trained on the software installed on Atlas VO (50,000 files) using 200 tags as described in Section 4.1. The classifier is then applied in all indexed files of Minersoft (1,062,154 files). Figure 8 presents the advanced search page of MinerSoft that provides the user with the ability to perform a tag based search as well as other options. Preliminary results have shown that there are some very frequent tags and the rest of the tags are assigned to a small number of files. However, as users assign more tags, the classifier will be regularly retrained and improve its predictive performance.

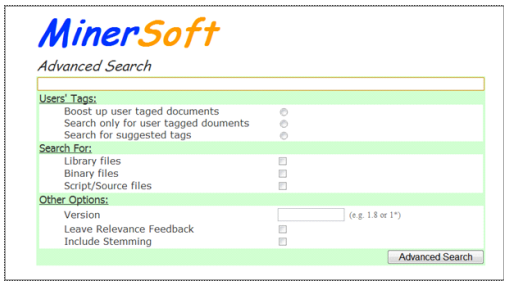

Figure 8. MinerSoft's advanced search page

7. The Utility of Tagging

In order to demonstrate the utility of tagging for the retrieval of software in Cloud infrastructures we have extracted projects from Google Code[2]. In Google Code, developers add tags to their projects. In Table 2 we present statistics regarding tagging behavior in projects that we retrieved from Google Code. We have installed these projects in a VM of our Cloud infrastructure[3]. Nephelae is a cloud computing cluster running Ubuntu Enterprise Cloud (UEC) - Ubuntu Server 11.04 LTS and open source Eucalyptus v2.0.

After applying MinerSoft's crawling and indexing procedures in the Google code data set, we observe that in the majority of the occasions, words that appear as tags do not appear in the files of the projects. Therefore, the keyword search in these cases would have failed since tags express more abstract and representative concepts of the project that they are assigned to. For example, `psvm` is a project installed in Nephelae Cloud (an implementation of parallel Support Vector Machine classification algorithm). In this case, we observe that important tags like "Machine-Learning", "Support-vector-machine", "Classifier" do not appear as words in the software files contained in the project and indexed by MinerSoft. Thus, without the tagging process, a search using these tags would fail.

Finally, the frequent usage of tags demonstrates the belief of developers that by tagging their projects they will make them easier to be retrieved. Figure 9 depicts tag frequency of each tag (for the sake of presentation we have not included tags that appear only once). We observe that the there is a small number of tags with high frequency while the majority present a small number of appearances following a long-tail distribution.

2. http://code.google.com/hosting/
3. http://grid.ucy.ac.cy/Nephelae/

Number of Projects	1208
Number of Tagged Projects	1160
Number of Unique Tags	3735
Average Tags per Project	6.03
Average Projects per Tag	1.95

Table 2. Tagging in projects from Google code

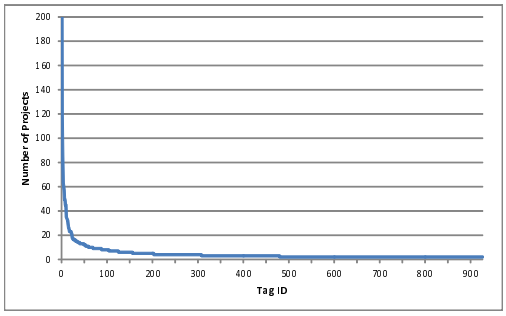

Figure 9. Tag Frequencies in Google Code Data

8. Conclusions

In this paper, we introduced tagging and automated-tagging capabilities into Minersoft, a software search engine for Cloud and Grid infrastructures. In order to provide automated tag assignments we formulated the problem as a machine learning multi-label classification task. Automated-tagging will improve the retrieval of software resources in Grid/Cloud infrastructures enabling users to easily located applications suitable to their needs.

Acknowledgments

This work was supported by the authors' Startup Grant, funded by the University of Cyprus.

References

[1] G. Pallis, A. Katsifodimos, and M. D. Dikaiakos, "Searching for software on the egee infrastructure," *J. Grid Comput.*, vol. 8, no. 2, pp. 281–304, 2010.

[2] A. Katsifodimos, G. Pallis, and M. D. Dikaiakos, "Harvesting large-scale grids for software resources," in *CCGRID*, F. Cappello, C.-L. Wang, and R. Buyya, Eds. IEEE Computer Society, 2009, pp. 252–259.

[3] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby, "A search engine for finding highly relevant applications," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 475–484.

[4] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, "Sourcerer: mining and searching internet-scale software repositories," *Data Min. Knowl. Discov.*, vol. 18, no. 2, pp. 300–336, 2009.

[5] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Trans. Softw. Eng.*, vol. 28, no. 10, pp. 970–983, 2002.

[6] T. Vanderlei and et. al., "A cooperative classification mechanism for search and retrieval software components," in *SAC '07*. New York, NY, USA: ACM, 2007, pp. 866–871.

[7] C. Treude and M.-A. Storey, "The implications of how we tag software artifacts: exploring different schemata and metadata for tags," in *Proceedings of the 1st Workshop on Web 2.0 for Software Engineering*, ser. Web2SE '10. New York: ACM, 2010, pp. 12–13.

[8] M.-A. Storey, J. Ryall, J. Singer, D. Myers, L.-T. Cheng, and M. Muller, "How software developers use tagging to support reminding and refinding," *IEEE Trans. Softw. Eng.*, vol. 35, pp. 470–483, July 2009.

[9] B. Sigurbjörnsson and R. van Zwol, "Flickr tag recommendation based on collective knowledge," in *Proceeding of the 17th international conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 327–336.

[10] L. Chen, L. Hu, Z. Zheng, J. Wu, J. Yin, Y. Li, and S. Deng, "Wtcluster: Utilizing tags for web services clustering," in *Service-Oriented Computing*, ser. Lecture Notes in Computer Science. Springer, 2011, vol. 7084, pp. 204–218.

[11] A. Byde, H. Wan, and S. Cayzer, "Personalized tag recommendations via tagging and content-based similarity metrics," in *Proceedings of the International Conference on Weblogs and Social Media*, March 2007.

[12] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," in *Data Mining and Knowledge Discovery Handbook*, 2nd ed., O. Maimon and L. Rokach, Eds. Springer, 2009.

[13] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern Recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

[14] M. Zhang and Z. Zhou, "Multi-label neural networks with applications to functional genomics and text categorization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, pp. 1338–1351, 2006.

[15] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Random k-labelsets for multilabel classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, pp. 1079–1089, 2011.

[16] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, "Mulan: A java library for multi-label learning," *Journal of Machine Learning Research*, 2011.