MUTAMA: An Automated Multi-label Tagging Approach for Software Libraries on Maven

Camilo Velázquez-Rodríguez, Coen De Roover
Software Languages Lab, Vrije Universiteit Brussel, Belgium
{camilo.ernesto.velazquez.rodriguez, coen.de.roover}@vub.ac.be

***Abstract*—Recent studies show that the Maven ecosystem alone already contains over 2 million library artefacts including their source code, byte code, and documentation. To help developers cope with this information, several websites overlay configurable views on the ecosystem. For instance, views in which similar libraries are grouped into categories or views showing all libraries that have been tagged with tags corresponding to coarse-grained library features. The MVNRepository overlay website offers both category-based and tag-based views. Unfortunately, several libraries have not been categorised or are missing relevant tags. Some initial approaches to the automated categorisation of Maven libraries have already been proposed. However, no such approach exists for the problem of tagging of libraries in a multi-label setting.**

This paper proposes MUTAMA, a multi-label classification approach to the Maven library tagging problem based on information extracted from the byte code of each library. We analysed 4 088 randomly selected libraries from the Maven software ecosystem. MUTAMA trains and deploys five multi-label classifiers using feature vectors obtained from class and method names of the tagged libraries. Our results indicate that classifiers based on ensemble methods achieve the best performances. Finally, we propose directions to follow in this area.

***Index Terms*—multi-label; libraries; software ecosystem; classification**

I. Introduction

Software products that evolve together in the same environment form a software ecosystem [1]. Examples include Maven[1], NPM[2] and CTAN[3], in which the co-evolving products are software libraries intended for reuse. The Maven ecosystem alone, intended for JVM-based libraries, is home to more than 2 million software products [2]. It can be challenging to find a suitable library to reuse in such a vast ecosystem.

To support the users of the Maven ecosystem, indexing platforms such as *Sonatype*[4] and *MVNRepository*[1] have been created. *Sonatype* supports searching for libraries based on the library's `GroupID`, `ArtifactID` or `Version`. *MVNRepository* in addition supports searching based on library categories and on library tags. Categories such as *Collections* group similar libraries from the same domain. Tags on *MVNRepository*, in contrast, are intended to correspond to the coarse-grained and possibly unique features of a library. Apache library *Commons-CLI*[5], for instance, has been tagged with the tags *command-line*, *cli* and *parser*. The library does indeed provide reusable functionality for parsing command line arguments. Unfortunately, not all libraries indexed by *MVNRepository* have been categorised and tagged this precisely. This is often the case for libraries that have only recently been contributed to the ecosystem or libraries that aren't enjoying widespread use. An automated approach to suggesting domain categories or feature tags for a software library could overcome this problem and thereby facilitate ecosystem search.

Linares-Vásquez et al. [3] have used single-label machine learning algorithms to predict which *SourceForge*[6] category a software product belongs to. While the approach could be transposed to libraries and software ecosystem indexing platforms, the single-label algorithms can only assign a single category which precludes their use for automated tagging.

Vargas-Baldrich et al. [4] approach library categorisation as multi-label problem instead. However, the approach uses TF-IDF instead of machine learning to automatically generate tags from code, rather than to learn and later predict human-defined tags.

In this paper, we propose MUTAMA, a multi-label machine learning approach for tagging Maven libraries. Like the two other approaches [3], [4], it analyses the byte code of the libraries that are to be tagged. The approach can be instantiated with any of five multi-label classifiers. We train and evaluate the performance of five such instantiations in this paper.

II. Background

Before sketching the design space for the multi-label classifiers with which MUTAMA has to be instantiated, we first describe the different types of datasets these classifiers can be trained on.

In general, classifiers can be trained on three types of datasets. The first type of dataset is called a binary dataset as it contains only two classes to predict. This explains why classifiers trained on binary datasets can be very effective. The second, more complex, type is called a multi-class dataset as it includes more than two classes to predict. Classifiers trained on multi-class datasets can still achieve good performance. For both types of datasets, classifiers only need to predict one class per instance in the dataset. The third, and most complex, type of dataset is called a multi-label datasets as it includes more than one class to predict per data instance.

[1]https://mvnrepository.com/
[2]https://www.npmjs.com
[3]https://www.ctan.org
[4]https://search.maven.org/
[5]https://mvnrepository.com/artifact/commons-cli/commons-cli
[6]https://sourceforge.net

In order to perform well, classifiers on a multi-label dataset need to fit several distinct data distributions which increases the complexity of both training and evaluation. Multi-label classifiers predict whether a class is present for a specific instance. Multi-target classifiers predict weights in addition for each of the classes, representing the relevance of each class for an instance.

A multi-label classifier suffices for our approach. Several multi-label classifiers have been proposed. Below we discuss the multi-label classifiers that performed best in an extensive survey and empirical comparison by Madjarov et al. [5]. Note that multi-label classifiers need to be instantiated with a base classifier. That is, a multi-label classifier cannot make predictions on its own without consulting a base classifier. The Support Vector Machine algorithm has, for instance, been used as a base classifier for multi-label classifiers. Sequential Minimal Optimization (SMO) [6] can be used to speed up the training of SVMs used to this end.

A. Binary Relevance Classifiers

The Binary Relevance (BR) classifier [7] trains one classifier per label to predict. It assigns one class for a label and another class for the rest of the labels. In other words, it simplifies the multi-label problem to a single-label one by binarising the dataset.

Like BR, Classifier Chaining (CC) [8] also uses n binary classifiers. The algorithm "chains" these classifiers together. Each binary classifier is assigned a label j the relevance of which will be learned. The compute the remaining relevance of the $j + q \,|\, q > 0$ labels, prior knowledge will be used.

B. Ensemble Method Classifiers

The RAndom *k*-LabELsets (RAkEL) ensemble classifier [9] forms small groups of labels and trains a single classifier for each group. It does consider relations among the labels within a group. Being an ensemble algorithm, predictions are made by majority vote from the classifiers for each group.

The Ensemble Multi-label (EML) classifier proposed by Read [10] combines several multi-label classifiers in an ensemble way. It can, for instance, be instantiated with the aforementioned Binary Relevance (BR) and Classifier Chaining (CC) classifiers.

III. MUTAMA APPROACH

We now describe the details of our approach to assigning tags for libraries in a software ecosystem. The approach is trained on a dataset of libraries that have already been tagged with two or more tags, corresponding to the coarse-grained functionality offered by the library. Section IV-A gathers such a dataset from the *MVNRepository* indexing platform for the Maven ecosystem.

Figure 1 depicts the steps comprising the training phase of the approach. The *Main Pipeline* box at the top of the figure depicts how a trained model is obtained by training a multi-label classifier on the corpus of tagged libraries. The first step

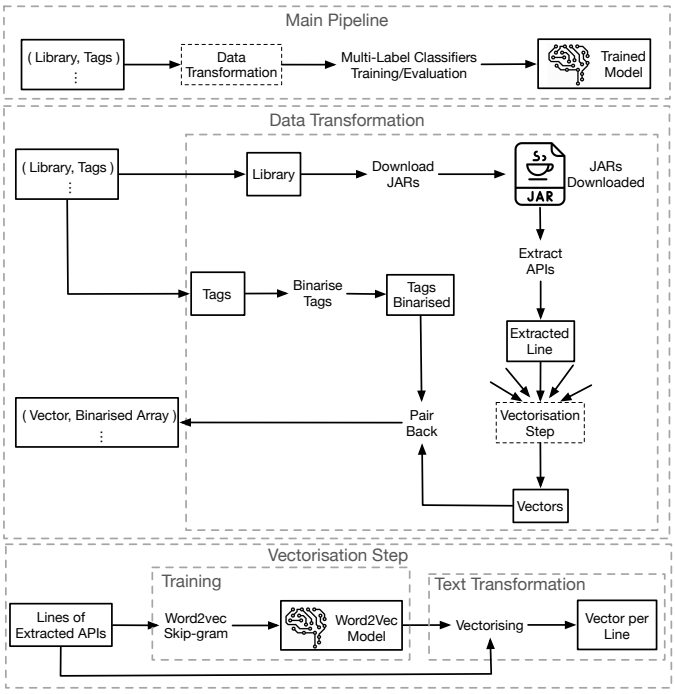

Fig. 1. The steps followed by tools following the MUTAMA approach.

entails transforming this group into a format that is suitable for such a classifier.

The *Data Transformation* step is depicted in the middle of the figure. For each library version, it takes as input a pair of a triplet `GroupID` - `ArtifactID` - `Version` and the set of associated tags. It then downloads the binary corresponding to each library version triplet. Next, the downloaded binaries are processed using the Byte Code Engineering Library (BCEL)[7] to extract the class and method names of their public APIs. This ensures that the machine learning classifier will learn patterns in the public interface of the libraries that have been tagged similarly, which is the same interface clients have access to. To help generalise the data, we split camel cased names into their constituent tokens. The resulting pieces of texts are appended into a single line per library version triplet.

Next, the extracted lines with textual information for all libraries are vectorised. Natural language processing approaches extract features from text using techniques such as Bag-of-Words [11], TF-IDF [12] and Word2Vec [13]. We use the latter due to its ability to recognise words that are semantically similar because they often occur in close contexts. Two neural network architectures can be used for WordVec. The Continuous-Bag-of-Words (CBOW) architecture predicts a target token from the context, whereas Skip-gram predicts context from a token. We selected the Skip-gram architecture for our approach as it has more learning capabilities than CBOW [14]. The vectorisation itself requires two steps, depicted at the bottom of Figure 1. The first training step takes a corpus of documents (i.e., the lines extracted for each of the libraries) and iterates

[7]https://commons.apache.org/proper/commons-bcel/

over a shallow neural network until a threshold or convergence is reached. The learned weights correspond to vectors for each of the unique tokens in the corpus. As a second step, we transform each document in the corpus by mapping the tokens at each line to their corresponding vectors. All vectors corresponding to tokens in a line are then averaged into a single vector per line.

Concurrently, in the middle of Figure 1, the set of tags associated with each line of library text is transformed into a binary array. For each library, MUTAMA constructs a zero-vector (i.e., a vector that only contain zeros) with a length equal to the total number of tags in the dataset. It then replaces zeros with ones at those positions corresponding to a tag of the library. In this way, the transformed dataset is ready to be used by multi-label machine learning classifiers.

Tying everything together, at the top of Figure 1, the Word2Vec vectors stemming from each library's byte code represent the input to a multi-label machine learning classifier. The binary arrays stemming from each library's tags on *MVNRepository* represent the classes to predict by the classifier. The result is a trained model which can be used to automatically tag untagged libraries in the Maven software ecosystem.

IV. Instantiation and Evaluation of MUTAMA

This section describes the evaluation of MUTAMA. The performance of MUTAMA depends on the multi-label classifier it is instantiated with, and on the quality of the dataset of tagged libraries this classifier is trained on. We will therefore first describe how we collected such a dataset from the *MVNRepository* website, before training several candidate classifiers on this dataset and comparing their performance.

A. Collecting a Corpus of Tagged Libraries

To collect a dataset of tagged libraries, we implemented one crawler to obtain their binaries and one crawler to obtain their tags.

The first crawler operates on the Maven ecosystem[8]. From this source we gathered the `GroupID - ArtifactID - Version` triplet for each library and its associated bytecode. We only consider the latest version of each library in this NIER paper. In total, we collected 235 011 Java and Scala libraries in this way. More versions could be collected, but this would increase the time for data extraction. For the remainder of the evaluation, we extracted statistically significant sample from this data. A confidence level of 99% and a confidence interval of $\pm$ 2 were used as parameters for the online Sample Size Calculator tool[9]. The result is a recommended sample size of 4 088 libraries.

The second crawler operates on the *MVNRepository* indexing platform[1]. Besides tags, categories and usage statistics are collected on this platform. Our crawler retrieves the tags for each of the 4 088 libraries in our aforementioned random sample. As expected, not all sampled libraries are tagged. Figure 2 depicts the distribution of the number of tags in our library sample. Only 3 137 libraries have been tagged with one or more tags, representing 77% of the sample. In the remainder of the evaluation, we will restrict the data to libraries carrying two or more tags, corresponding to the 59% of libraries with at least one tag. As we will illustrate below, we do not deem a single tag sufficient to facilitate searching through an ecosystem nor for training an automated classifier.

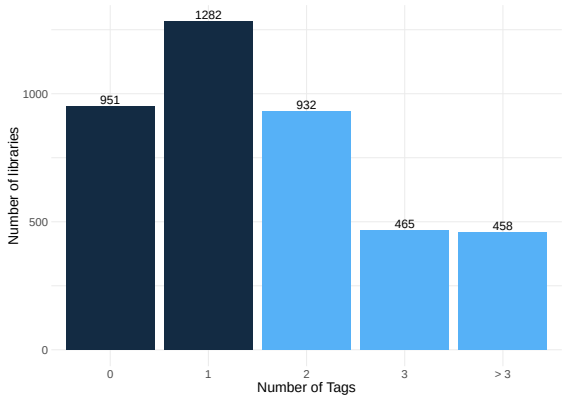

Fig. 2. Distribution of the number of tags for the 4 088 sampled libraries.

B. Scarcity of Tags as a Motivation for Automated Tagging

We discuss two example libraries tagged with the tag *scala*. Library *Tokenizers*[10] only carries the *scala*, and does not carry a tag corresponding to its code analysis functionality nor a tag corresponding to its tokenisation functionality. Although this kind of information could be inferred from its `GroupID` or `ArtifactID`, the library will only appear in the results of ecosystem searches for the tag *scala*. Library *RL Expander*[11] carries the tag *web-framework* in addition to *scala*. The library expands URLs that appear shortened. Additional tags could be added to enrich the knowledge about this library within the software ecosystem. Examples of libraries with a good number of tags are *Mockito Scala*[12] and *Util Module for SBT*[13] which carry the tags <*scala mock scalaz*> and <*scala sbt build build-system*> respectively.

C. Instantiating MUTAMA with Multi-label Classifiers

We use the open source MEKA[14] tool to instantiate our MUTAMA approach with a multi-label classifier and its base classifier as explained in Section III. The MEKA tool provides all multi-label classifiers and base classifiers discussed in Section II.

Table I depicts the results for the different instantiations of MUTAMA on our dataset. Results were obtained using 10-fold cross-validation, where nine folds are considered for training and one for evaluation. The metrics for each of the folds are

[8]https://repo1.maven.org/maven2/

[9]https://www.surveysystem.com/sscalc.htm

[10]https://mvnrepository.com/artifact/org.scalameta/tokenizers_2.12

[11]https://mvnrepository.com/artifact/org.scalatra.rl/rl-expander_2.9.0

[12]https://mvnrepository.com/artifact/org.mockito/mockito-scala-scalaz_2.13.0-RC3

[13]https://mvnrepository.com/artifact/org.scala-sbt/util-collection_2.10

[14]https://sourceforge.net/projects/meka/

TABLE I
PERFORMANCE METRICS OF MUTAMA INSTANTIATED WITH DIFFERENT MULTI-LABEL CLASSIFIERS. THE BEST RESULTS FOR EACH METRIC ARE HIGHLIGHTED IN BOLD.

Classifier	2 Tags				3 Tags				>3 Tags			
	A	F1-M	F1-m	HL	A	F1-M	F1-m	HL	A	F1-M	F1-m	HL
BR	0.01	0.01	0.03	**0.05**	0.07	0.08	0.16	**0.08**	0.19	0.21	0.42	**0.06**
CC	0.02	0.03	0.06	**0.05**	0.1	0.11	0.18	0.09	0.21	0.23	0.45	**0.06**
RaKel	0.01	0.01	0.03	**0.05**	0.08	0.1	0.16	0.09	0.15	0.18	0.35	0.07
EML(BR)	0.04	0.05	0.08	**0.05**	0.12	0.13	0.24	0.09	0.24	0.24	**0.47**	0.07
EML(CC)	**0.24**	**0.29**	**0.32**	0.09	**0.17**	**0.21**	**0.29**	0.12	**0.27**	**0.31**	0.46	0.09

later averaged to get the numbers depicted in the table. The table is divided into three regions for the analysis of libraries which carry two, three and more than three tags. As can be seen from the table, the results differ between the regions.

The metrics considered in the evaluation are *Accuracy* (A), *F1-score macro* (F1-M), *F1-score micro*(F1-m) and *Hamming loss* (HL). F1-score macro refers to the average of the F1 scores per predicted tag, that is, it reflects the quality of the individual predicted tags. F1-score micro score is computed by considering all tags in the dataset. Hamming Loss is the ratio of incorrectly predicted tags to the total number of tags. This loss function should be optimised by classifiers, zero being its optimal value.

The results for libraries with only two tags are low for all classifiers but EML in combination with base classifier CC. There is, moreover, a noticeable gap between the F1 micro score for the best (0.32) and the second-best classifier (0.08). The same can be said about the results for the F1 macro metric. This reflects the effectiveness of ensemble algorithms in general, and the classifier chain ones in particular. Surprisingly, the classifier scoring best on accuracy, F1 micro, and F1 macro scores the lowest on the Hamming loss metric. However, the difference of 0.04 can be considered negligible.

The results for the libraries that carry exactly three tags are similar. The ensemble classifier EML(CC) once more obtains the best scores for the accuracy, F1 micro, and F1 macro metrics but the worst score for the Hamming loss metric. The gap between this classifier and the others is again noticeable. The other ensemble classifier EML(BR) consistently ranks as the second best. Interestingly, the Binary Relevance classifier achieves the worst accuracy but scores the best on the Hamming loss metric.

The results for libraries that carry more than three tags in the ground truth are interesting. We had expected the classifiers to score lower than before as more tags need to be predicted. In reality, their performance is better than before. The two ensemble classifiers once again achieve the best scores.

D. Discussion of Results

The differences in performance of the classifiers on libraries with 2, 3, or more than three tags could stem from differences in their respective distribution in the dataset. These differences are apparent from Figure 2, which depicts the number of libraries in each group in a lighter shade of blue. More importantly, each group is imbalanced as some tags are simply more prevalent on the *MVNRepository* indexing platform. The use of traditional cross-validation in our evaluation does not help in this case, as it does not consider data distributions in its assignment of instances to folds. Although some stratification techniques have been proposed to address imbalanced datasets for multi-label problems [15], they are not yet available in the MEKA tool suite which we used to instantiate MUTAMA. We therefore consider the use of stratified cross-validation for the training and evaluation of the MUTAMA classifiers as future work.

TABLE II
PREDICTIONS MADE BY THE BEST MODEL.

Library	**True tags**	**Predicted tags**
Akka HTTP Backend	distributed, actor, akka concurrency, client, http	distributed, actor, akka concurrency, http
AWS SDK Scala.js Facade QuickSight	scala, aws, amazon scalajs, sdk	scala, aws, amazon scalajs, sdk
TestNG Interface	io, testing	testing
Camel Labs IoT Components Device IO	github, io	-
Gradle Code Quality Tools Plugin	tools, build, build-system plugin, groovy, gradle	-

For five randomly selected libraries, Table II depicts the ground truth and the predictions made by the best performing EML(CC) classifier. The prediction for library *Akka HTTP Backend* in the first row is only missing the *client* tag. As this library focuses on the back end, a *server* tag might have been more appropriate both in the ground truth and in the prediction. In the second row, all tags in the ground truth for library *AWS SDK Scala.js* have been predicted. For the third row, the classifier failed to predict that library *TestNG* is somehow linked to *io* operations. Again, this failure might as well be attributed to mistakes or unexpected tags in the data on the *MVNRepository* indexing platform. The tags on the platform are, after all, maintained by volunteers who might wrongly assign a tag or miss appropriate ones. Finally, for the libraries on the last two rows, the classifier was unable to predict a single tag.

We believe that such failures could be caused by several reasons. First, the number of libraries for each combination of tags is small. Machine learning classifiers train better with a higher number of instances from which they can extract patterns. Relevant patterns between library APIs and combinations of tags may therefore remain unlearned. Second, the API information extracted from the binaries might not be representative enough. Although hidden from users, informa-

tion that is relevant for tagging purposes might be hiding in the implementation of that API. Finally, word embeddings of class and method names might not sufficiently capture the functionality of a library. Alternatives such as the ASTs of their source code, traces of their static symbolic execution, or execution logs on Travis CI[15] could be explored as complementary sources of information in the future.

V. Related Work

We discuss prior work on multi-label classification and software categorization.

A. Applications of Multi-label Classifiers

Madjarov et al. [5] empirically compare several multi-label classifiers implemented in MEKA, MULAN[16] and CLUS[17]. Their comparison on 16 metrics includes 12 multi-label classifiers combined with the SVM, Decision Tree and Nearest Neighbor base classifiers on 11 datasets from different domains. Ensemble classifiers performed best overall, in particular when combined with the SVM base classifier. We based our initial selection of classifiers on this empirical comparison. However, we limited our scope to those algorithms provided by the MEKA tool and used SMO to speed up the training of SVMs.

Liu and Chen [16] propose a multi-label approach for sentiment analysis of microblogs. They use most of the classifiers described in Section II. In this application too, ensemble algorithms outperform the other multi-label classifiers. Like MUTAMA, the approach uses text segmentation, feature extraction, and multi-label classification. Besides these similarities do exist, the features used by MUTAMA are different as it was not intended for sentiment analysis.

B. Software Categorisation

Linares-Vásquez et al. [3] employ machine learning techniques to classify software into given categories. They extract the public API of third-party libraries and the *SourceForge* category to which they have been classified. This information is used to train five machine learning algorithms: SVM, Naive Bayes, Decision Trees, RIPPER and IBK. Here too, SVM achieved the best results. Our approach use multi-label instead of single-label classifiers, and uses different word embeddings.

VI. Conclusion and Future Work

Categories and tags facilitate the search through vast software ecosystems such as Maven. We have shown that a statistically significant sample of *Maven* libraries misses such tags on the accompanying *MVNRepository* indexing platform. Their usefulness to ecosystem users may be limited as many tagged libraries only carry a single tag.

We propose MUTAMA, an automated multi-label classification approach that suggests *MVNRepository* tags for libraries in the *Maven* software ecosystem. The approach operates on the binary of a library, from which it extracts the class and method names in the public API. MUTAMA needs to be instantiated with a multi-label classifier, which is trained on embeddings of the extracted API names. The results show that two ensemble classifiers outperform the three other classifiers, aligning with the findings of previous work on multi-label classification problems.

For future work, we consider a more extensive evaluation with multi-label classification algorithms beyond those included in the MEKA toolkit used in our implementation.

[15]https://travis-ci.org/

[16]http://mulan.sourceforge.net/

[17]http://clus.sourceforge.net

Acknowledgements

This research was partially funded by the Excellence of Science project 30446992 SECO-Assist financed by FWO-Vlaanderen and F.R.S.-FNRS.

References

[1] M. Lungu, "Towards reverse engineering software ecosystems," in *2008 IEEE International Conference on Software Maintenance*. IEEE, 2008, pp. 428–431.

[2] A. Benelallam, N. Harrand, C. Soto-Valero, B. Baudry, and O. Barais, "The maven dependency graph: a temporal graph-based representation of maven central," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 344–348.

[3] M. Linares-Vásquez, C. McMillan, D. Poshyvanyk, and M. Grechanik, "On using machine learning to automatically classify software applications into domain categories," *Empirical Software Engineering*, vol. 19, no. 3, pp. 582–618, 2014.

[4] S. Vargas-Baldrich, M. Linares-Vásquez, and D. Poshyvanyk, "Automated tagging of software projects using bytecode and dependencies (n)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 289–294.

[5] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Džeroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern recognition*, vol. 45, no. 9, pp. 3084–3104, 2012.

[6] J. C. Platt, "Advances in kernel methods. chapter fast training of support vector machines using sequential minimal optimization," *MIT Press, Cambridge, MA, USA*, vol. 3, pp. 185–208, 1999.

[7] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 3, no. 3, pp. 1–13, 2007.

[8] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Machine learning*, vol. 85, no. 3, p. 333, 2011.

[9] G. Tsoumakas and I. Vlahavas, "Random k-labelsets: An ensemble method for multilabel classification," in *European conference on machine learning*. Springer, 2007, pp. 406–417.

[10] J. Read, "Scalable multi-label classification," Ph.D. dissertation, University of Waikato, 2010.

[11] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 43–52, 2010.

[12] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, 1972.

[13] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[15] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "Addressing imbalance in multilabel classification: Measures and random resampling algorithms," *Neurocomputing*, vol. 163, pp. 3–16, 2015.

[16] S. M. Liu and J.-H. Chen, "A multi-label classification based approach for sentiment classification," *Expert Systems with Applications*, vol. 42, no. 3, pp. 1083–1093, 2015.