Machine Learning on Android with Oracle Tribuo, SMILE and Weka[*]

Máté Szabó

Faculty of Informatics, University of Debrecen, Hungary
`szabo.mate@inf.unideb.hu`

Proceedings of the 1st Conference on Information Technology and Data Science
Debrecen, Hungary, November 6–8, 2020
published at http://ceur-ws.org

Abstract

Machine learning is reaching nearly every programming language and most kinds of devices. While the most popular language for developing machine learning application is Python, it has its own limits, for example, the partial compatibility with Android devices. When a mobile application needs to train a model, it is easier to achieve this with the device's native language like Java or Kotlin. There are many machine learning libraries for Java, but most of them lack Android support. This paper compares the resources needed to train random forest, support-vector machine and K-means models of the Weka, Tribuo and SMILE libraries. We developed an application to compare these libraries' implementations on datasets with various sizes. The results show that Weka is the suggested library for bigger datasets and complex models, as it is the least resource hungry.

Keywords: Android, machine learning, Tribuo, SMILE, mobile

1. Introduction

In September 2020, Oracle announced Tribuo, their open source Java machine learning library under Apache 2.0 license. It features many commonly used algo-

[*]The work is supported by the EFOP-3.6.1-16-2016-00022 project. The project is co-financed by the European Union and the European Social Fund.

rithms like random forest, SVM, lasso, K-means, so it can solve prediction, classification, regression, clustering and anomaly detection problems. SMILE, the Statistical Machine Intelligence and Learning Engine is another machine learning library for Java. Its main advantage is performance compared to other libraries and algorithm support. Weka is a general purpose open source machine learning software with Java API, which is easy to use and has its own graphical interface. The common thing in these libraries is that they can be used with Java or Kotlin and there are many algorithms that all of them support. Because of this, their performance can be compared in the same environment, which will be an Android device with a mobile processor in it. Although these are not native Android libraries, they can work on these systems and their performance can be compared. Kotlin is a programming language for JVM, which became the preferred language for Android programming. Codes from Java can be transformed into Kotlin code, so it is easy to use Java libraries in this environment. Benchmarking mobile devices' model training performance is a repeating task, because we can measure how much these devices evolved in years. In this paper, we present the Android machine learning ecosystem, the libraries, the challenge of porting machine learning libraries, and the results.

2. Related Works

With the evolution of mobile devices and applications, it was inevitable to use machine learning techniques for more personal user experience. Most applications use pre-trained models to recognize voice, to take better pictures or to swap faces. There are many disadvantages of training models on mobile, for example, the energy consumption [14]. Besides that, there are many use cases of models trained on mobiles like comparison of machine learning capability of processors [8], detecting potholes [9], or malware [19]. The present work can be placed on the topic of machine learning and Android benchmarking. There are many articles about comparing devices or machine learning libraries by training time, memory and CPU efficiency. For a complete benchmarking tool, there is the PMLB, the Penn machine learning benchmark [16]. There are other papers about benchmarking, like the Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark [4], the MLPerf Training Benchmark [12], or the Benchmark of Machine Learning Methods for Classification of a Sentinel-2 Image [18].

2.1. Java Machine Learning

Java is not a popular language for machine learning, but it is popular for application development, and because of the need of intelligent applications, it supports many machine learning features with libraries. Each of these supports different algorithms and datasets and each of them has advantages for specific systems. Weka [6] is a complete machine learning software with graphical interface, command-line interface and it can be used as a Java library too. It supports tasks like pre-processing, classification, regression, clustering, association rules and visualization. SMILE [10], the Statistical Machine Intelligence and Learning Engine is a powerful engine that covers every aspect of machine learning. It supports JVM languages, so SMILE codes can be transformed into Kotlin codes. It has many algorithms for classification, regression, clustering, association rule mining and many built-in solutions for pre-processing, validation, feature engineering and time series. Oracle Tribuo [17] is a newly open sourced machine learning library written in Java and it has unique features like provenance, type safety and interoperability. Models, datasets, and evaluations have provenance, which means they know the transformations and parameters used to create them. The interoperability means that Tribuo has interfaces to libraries like XGBoost [3] and Tensorflow [1] and the ONNX [2] exchange format. Deeplearning4J [21] is a SMILE based library focusing on deep learning, which is not in the scope of this paper. H2O [7] is an in-memory platform that has many softwares and libraries for machine learning. It supports most kind of data mining tasks and it can be integrated with Java applications through REST API or embedding. Mallet [13] is another option for applications which use natural language processing, document classification or clustering.

3. Machine Learning on Android

When we want to train and use machine learning models on Android, we can encounter many challenges. The main reason not to train models especially with large datasets on mobiles is that these operations need a lot of energy, so running applications would result in battery drain. While training is possible, it is limited by these devices' memory capacity, because most mid-range smartphones usually have 4-6 gigabytes of memory, and this is not enough for processing larger datasets, not to mention that Android has limitations for memory usage. Another challenge for mobile machine learning developers is the architectural difference between the processors of computers and mobiles. Using Java libraries on these devices can be risky because Android does not have full Java support, so it is possible that some functions don not work or work, but with different results.

Currently, applications that want to use machine learning can choose from using TensorFlow Lite, or a library with Android Neural Network API support, or web services. With TensorFlow Lite developers can mostly use pre-trained models created with TensorFlow or they can train specific models for image and text classification. A popular choice is to use machine learning web services where the application sends data to the service and gets back the result.

3.1. Porting Weka

Weka contains many Android incompatible code, mostly its graphical interface and logging, but there are many more functions that use specific code parts, which cannot be compiled on mobiles. Our strategy here was to remove everything that is incompatible and see how the application can work with the newly compiled Weka.

The porting was successful, however sometimes minor errors occured during the tests. The tested library was based on rjmarsan's Weka-for-Android project [11].

3.2. Porting SMILE

SMILE also contained incompatible code, for example Java codes that were not supported by Android or functions with `java.sql` type. Because the number of these code parts were few, they were replaced by Android compatible ones. The result was good enough to run on mobiles, but it did not support all features, and some of these errors occured during runtime.

3.3. Porting Tribuo

Oracle's Tribuo library consists of many Maven artifacts, and some of them were not needed for this application. There are modules which cannot be compiled on Android, but the most important ones, namely tribuo-data, tribuo-classification-trees, tribuo-clustering-kmeans and tribuo-classification-sgd ran without modifications. Overally we can say that not all of Tribuo's functions work on Android, especially the ones using third-party libraries, but we can do simple machine learning tasks with it on mobiles.

4. Application

The aim of the Android application is to measure properties of machine learning libraries like memory usage or runtime. The libraries involved are the newly open sourced Oracle Tribuo, SMILE and Weka. The application runs the same test with each library, which means it trains models like SVM, Random Forest and K-means on multiple datasets with different sizes. The test uses the same algorithms and parameters for all libraries. The results and runtime properties are logged by the application.

The graphical interface is quite simple. When we tap on a library name, the software will train a selected type of model on a selected dataset. The "ALL" button is for running the training operation of all libraries parallelly. We can choose from the Iris dataset [5] with 150 records, a subset of the Record Linkage Comparison Patterns dataset [15, 20] with 60,000 records and the full Record Linkage Comparison Patterns dataset with 5749132 records. The algorithms we can choose from are the random forest with 500 trees, split rule GINI, maximum depth = 20, maximum nodes = data size / 5 and node size = 5, the SVM with an RBF kernel, gamma = 0.1, lambda = 0.5, epochs = 30, and the K-Means algorithm with 2 or 3 centroids based on the dataset, iterations = 10 and distance is Euclidean. The outputted result contains the runtime, the maximum and the average CPU usage, the memory usage and the energy consumption.

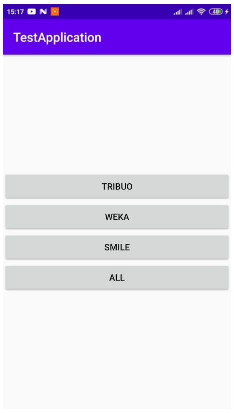

Figure 1. First screen of the measuring application. The user can decide to run tests for a specific library or all available libraries.

5. Results

Weka, SMILE and Tribuo can train machine learning models on Android and all of them can be used for simpler applications. As the software trained the same models with the same parameters and datasets these models' performance was the same on the test datasets. The only difference between these libraries was in the runtime, memory usage, average CPU usage and energy consumption.

5.1. Runtime

In Table 1 and Figure 2, we can see how fast the libraries finished the training of the models, where the prefix is the dataset (i is the Iris dataset and p is the Patterns dataset), and the second part is the name of the algorithm (where r is the random forest, s is the support-vector machine and k is the K-means).

Table 1. Runtime of algorithms on specified datasets.

	i-r	i-s	i-k	p-6-r	p-6-s	p-6-k	p-r	p-s	p-k
Tribuo	0.98	1.18	0.45	49.72	0	25.29	3187.497	0	0
Weka	0.72	0.7	0.72	9.26	95.8	22.73	459.304	1696.58	92.817
SMILE	3.09	1.47	0.09	75.6	0	13.24	1725	0	0

As we can see, most of the time, Weka library was the fastest, except for K-means where SMILE was faster. For smaller datasets, both Tribuo's and SMILE's results are good, but for larger ones, they were much slower than Weka. Where there are zeros in the table, the software did not complete the training of the model, because the Android system killed the application due to its high resource need.

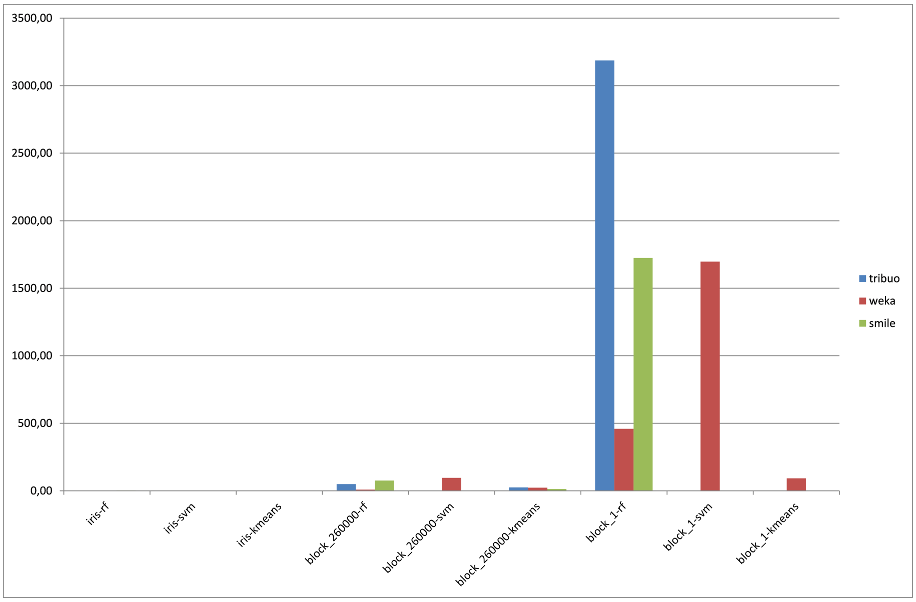

Figure 2. Comparison diagram of runtime results.

5.2. Memory Consumption

In Table 2 and Figure 3, we can see how much memory the algorithms needed in megabytes. The notation is the same as in Table 1 and Figure 2.

Table 2. Memory need of algorithms on specified datasets.

	i-r	i-s	i-k	p-6-r	p-6-s	p-6-k	p-r	p-s	p-k
Tribuo	110	95	102	106.3	500	139	500	500	290
Weka	82.5	103.4	96	89.3	95	133.8	316	354.6	330
SMILE	90.5	113	84	121.7	800	265.8	285.4	800	1200

It is clear, that Weka used the least amount of memory, but in some cases Tribuo was wery close to it, like in iris-svm, iris-kmeans, and patterns-kmeans. There are special cases where the training did not finish, like Tribuo's pattern-60000-svm, pattern-rsvm and pattern-kmeans or SMILE's pattern-60000-svm, pattern-svm and pattern-kmeans where the memory need was extremely high compared to other cases. The highest value was the SMILE's K-means training for the Pattern dataset where the application used 1.2 GB memory. For smaller datasets we can say that Tribuo and SMILE needed somewhat more memory, but this is not a huge difference.

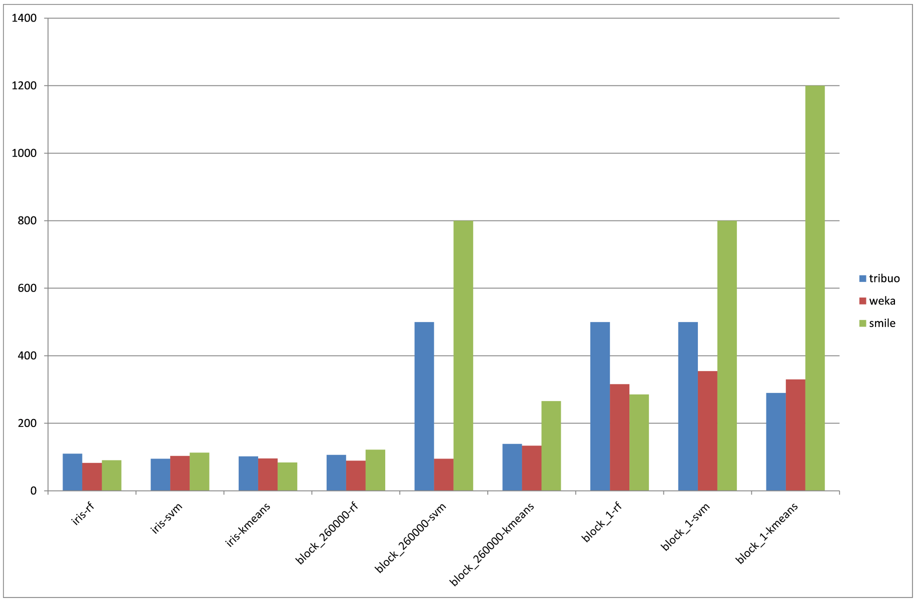

Figure 3. Comparison diagram of memory results.

5.3. Battery Consumption

In Table 3 and Figure 4 we can see the battery consumption of the application, when it trains the selected models. The table's and figure's notations are the same as earlier. The values range from 0 to 1, where 0 is the no energy need and 1 is the highest energy need.

Table 3. Energy consumption of algorithms on specified datasets.

	i-r	i-s	i-k	p-6-r	p-6-s	p-6-k	p-r	p-s	p-k
Tribuo	0.6	0.2	0.2	0.6	1	0.2	0.4	1	0.6
Weka	0.2	0.2	0.2	0.6	0.6	0.4	0.4	0.2	0.4
SMILE	0.6	0.2	0.6	1	0.6	0.4	1	0.6	0.2

Battery consumption is an important part of these measurements, because this means that a machine learning application could be maintained or it drains the battery that much, that the application is unusable. In Figure 4 we can see that Weka used the least amount of battery, the next one was Tribuo and the hungriest library was SMILE. For smaller tasks, Tribuo's and Weka's energy need were nearly the same.

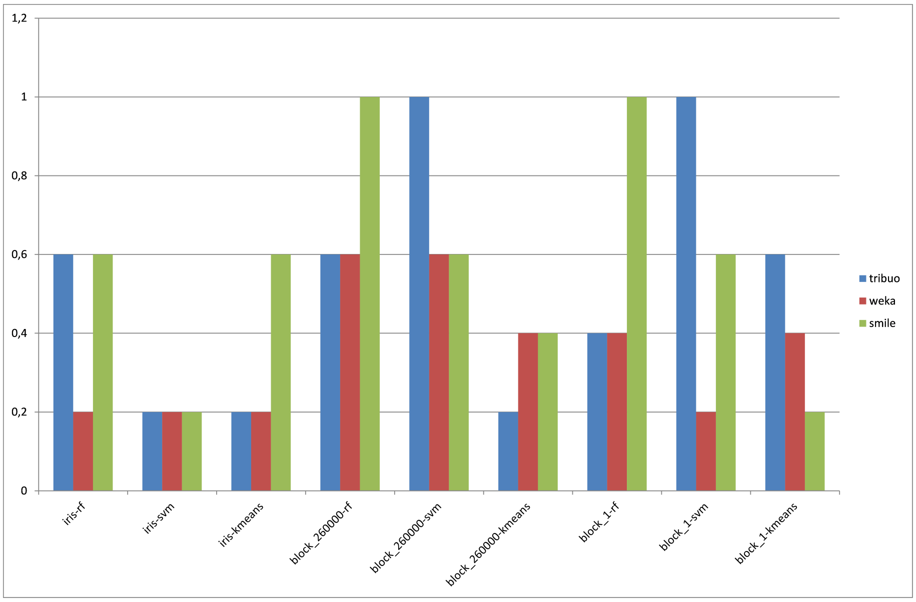

Figure 4. Comparison diagram of energy consumption results.

5.4. Average CPU Usage

In Table 4 and Figure 5 we can see the average CPU usage while training the models. The table's and figure's notations are the same as earlier. Average CPU usage shows how these libraries use this resource. When this value is below 50, it means that other applications can run parallelly while the training is running.

Table 4. CPU usage of algorithms on specified datasets.

	i-r	i-s	i-k	p-6-r	p-6-s	p-6-k	p-r	p-s	p-k
Tribuo	26	23	37	15	0	16	25	25	60
Weka	20	21	21	12	20	21	18	23	20
SMILE	97	71	16	90	30	21	75	30	61

Table 4 shows that SMILE needs the most CPU resource for every algorithm and dataset. The second most processor hungry library was Tribuo, but it produced similar results as Weka. This means that with better processors, SMILE would perform better in runtime. For weaker processors, it is strongly advised to use Weka because of its small resource need.

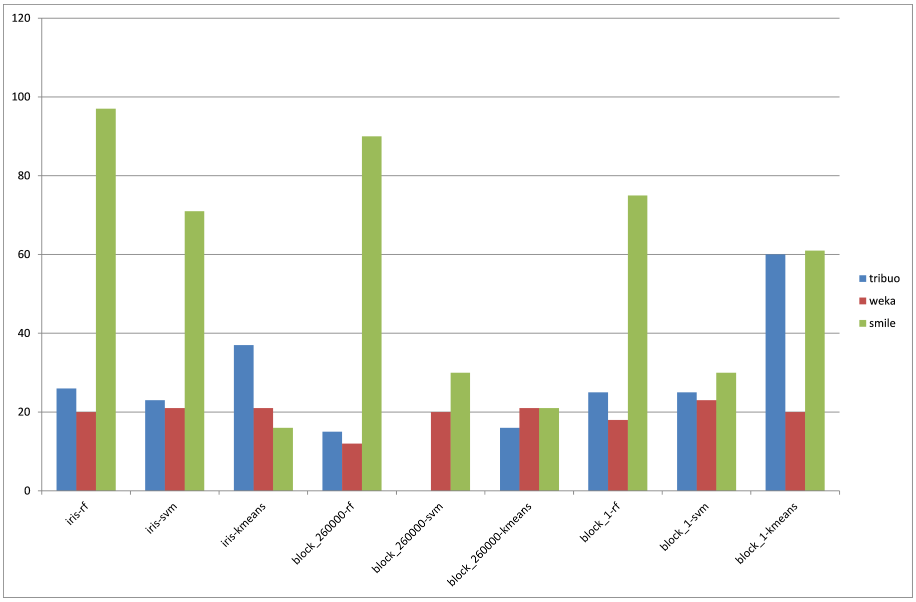

Figure 5. Comparison diagram of CPU usage results.

6. Conclusion

This paper showed that using machine learning libraries meant for Java can be used in Android application development if we have a specific task. However if an application uses models for image or text classification the recommendation is to use Tensorflow or web services. Specific tasks can be classifying or recommendation based on the users' data, where it is necessary to train a local model.

Porting Weka, Tribuo and SMILE to Android devices is a somewhat challenging task, because each of them has some Java version or platform specific codes, which will not work on mobiles. To compare these libraries' performance, we must select algorithms that each of them supports and has Android compatible implementation. For the comparison, we chose the random forest, SVM and K-means models and datasets with different sizes. The results show that Weka is the suggested library for bigger datasets and complex models, as it is the least resource hungry. It supports a wide range of algorithms, so every kind of machine learning task can be done with it. For smaller datasets or fewer complex models, Tribuo and Smile can be an option because they get updates frequently, so they can react faster to market needs.

An improvement can be extending these tests with other datasets, algorithms, or multiple devices. The results could be compared to results from computers instead of Android devices. As new machine learning libraries are released, they could be ported to mobiles and the results could be extended.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al.: *Tensorflow: A system for large-scale machine learning*, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16), 2016, pp. 265–283.

[2] J. Bai, F. Lu, K. Zhang, et al.: *ONNX: Open Neural Network Exchange*, https://github.com/onnx/onnx, 2019.

[3] T. Chen, C. Guestrin: *XGBoost: A Scalable Tree Boosting System*, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, San Francisco, California, USA: ACM, 2016, pp. 785–794, isbn: 978-1-4503-4232-2, doi: 10.1145/2939672.2939785,
url: http://doi.acm.org/10.1145/2939672.2939785.

[4] C. Coleman, D. Kang, D. Narayanan, L. Nardi, T. Zhao, J. Zhang, P. Bailis, K. Olukotun, C. Ré, M. Zaharia: *Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark*, ACM SIGOPS Operating Systems Review 53.1 (2019), pp. 14–25,
doi: 10.1145/3352020.3352024.

[5] R. Fischer: *Iris Dataset*, 1988,
url: https://archive.ics.uci.edu/ml/datasets/iris.

[6] E. Frank, M. A. Hall, I. Witten: *The WEKA workbench. Online appendix*, in: Data mining: practical machine learning tools and techniques, Morgan Kaufmann, 2016.

[7] H2O.ai: *H2O*, 3.10.0.8, Nov. 2020,
url: https://github.com/h2oai/h2o-3.

[8] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, L. Van Gool: *AI Benchmark: Running Deep Neural Networks on Android Smartphones*, in: Proceedings of the European Conference on Computer Vision (ECCV) Workshops, Sept. 2018,
doi: 10.1007/978-3-030-11021-5_19.

[9] A. Kulkarni, N. Mhalgi, S. Gurnani, N. Giri: *Pothole detection system using machine learning on Android*, International Journal of Emerging Technology and Advanced Engineering 4.7 (2014), pp. 360–364.

[10] H. Li: *Smile*, https://haifengl.github.io, 2014.

[11] R. Marsan: *Weka-for-Android*, https://github.com/rjmarsan/Weka-for-Android, 2011.

[12] P. Mattson, C. Cheng, G. Diamos, C. Coleman, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, D. Brooks, D. Chen, D. Dutta, U. Gupta, K. Hazelwood, A. Hock, X. Huang, D. Kang, D. Kanter, N. Kumar, J. Liao, D. Narayanan, T. Oguntebi, G. Pekhimenko, L. Pentecost, V. Janapa Reddi, T. Robie, T. St John, C.-J. Wu, L. Xu, C. Young, M. Zaharia: *MLPerf Training Benchmark*, in: Proceedings of Machine Learning and Systems, ed. by I. Dhillon, D. Papailiopoulos, V. Sze, vol. 2, 2020, pp. 336–349,
url: https://proceedings.mlsys.org/paper/2020/file/02522a2b2726fb0a03bb19f2d8d9524d-Paper.pdf.

[13] A. K. McCallum: *MALLET: A Machine Learning for Language Toolkit*, 2002,
url: http://mallet.cs.umass.edu.

[14] A. McIntosh, A. Hindle, S. Hassan: *What can Android mobile app developers do about the energy consumption of machine learning?*, Empirical Software Engineering 24.1 (2019), pp. 562–601,
doi: 10.1007/s10664-018-9629-2.

[15] E. C. R. of North Rhine-Westphalia: *Record Linkage Comparison Patterns Dataset*, 2011,
url: https://archive.ics.uci.edu/ml/datasets/record+linkage+comparison+patterns.

[16] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, J. H. Moore: *PMLB: a large benchmark suite for machine learning evaluation and comparison*, BioData mining 10.1 (2017), pp. 1–13,
DOI: 10.1186/s13040-017-0154-4.

[17] Oracle: *Oracle Tribuo*, https://tribuo.org/, 2020.

[18] F. Pirotti, F. Sunar, M. Piragnolo: *Benchmark of Machine Learning Methods for Classification of a SENTINEL-2 Image*, International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences 41 (2016),
DOI: 10.5194/isprsarchives-XLI-B7-335-2016.

[19] J. Sahs, L. Khan: *A Machine Learning Approach to Android Malware Detection*, in: 2012 European Intelligence and Security Informatics Conference, 2012, pp. 141–147,
DOI: 10.1109/EISIC.2012.34.

[20] M. Sariyar, A. Borg, K. Pommerening: *Controlling false match rates in record linkage using extreme value theory*, Journal of Biomedical Informatics 44.4 (2011), pp. 648–654,
DOI: 10.1016/j.jbi.2011.02.008.

[21] E. D. D. Team: *Deeplearning4j: Open-source distributed deep learning for the JVM*, 2016,
URL: http://deeplearning4j.org/.