JMLP: A Machine Learning Tool for Students and Instructors

B Nitish
Department of Computer Science
State University of New York at Binghamton
Binghamton, US
nbhanup1@binghamton.edu

Arpita Chakraborty
Department Of Computer Science
State University of New York at Binghamton
Binghamton, US
achakra4@binghamton.edu

***Abstract*—This Research to Practice Work in Progress paper presents a tool which is extremely useful in the domain of Machine Learning. Machine Learning is one of the emerging topics and sought-after in the field of computer science. Most of the universities and online educational platforms (MOOCs) offer courses related to this topic. Since most parts of the topic require students to be well versed in mathematics, particularly probability and the algorithms involve initial training before validating the results, it becomes difficult for a beginner student to program and test his understanding of the algorithm. Our tool solves this problem by providing functionalities in two ways: (1) we propose a novel way for students to learn and understand the concepts of machine learning and get feedback on their implementation of the algorithm, and (2) we provide instructors the statistics of the performance of the students, such as common mistakes committed, which helps them tailor their classrooms, making them more robust. Due to these functionalities, this tool can be integrated with online classrooms (MOOCs) enabling students to practice the online course materials, get personalized feedbacks and enable the authors of the course to conduct tests for a diverse ecosystem.**

Keywords—machine leaning tool, JMLP, MOOCs, automatic personalized feedback

I. Introduction

Machine Learning is emerging as an important topic in the domain of Computer Science. Introductory and advanced courses on this topic are offered at undergraduate and graduate levels. Many tools in this domain have been developed, such as WEKA [1], Scikit-learn [2] amongst others. They provide built-in implementations of various Machine Learning algorithms and provide application programming interfaces (APIs) which can be used on the required data. But these tools, for a beginner student learning these algorithms, does not provide much to aid their learning.

Besides, in diverse and large classrooms such as online classrooms (Massively Open Online Classrooms - MOOCs), manual evaluation of student solutions is near impossible. This requires an automated mechanism to evaluate and provide relevant feedback to the students upon their correctness, or otherwise, of their implementations. In this paper, we present our tool which provides functionalities for both instructors and students alike and is freely available at https://bit.ly/javamlproj with a comprehensive documentation on the installation and usage of the tool.

Machine Learning algorithms are mainly comprised of statistics - probabilities, more specifically. Most of the curricula involve students to implement a classification algorithm such as Naive Bayes, Logistic Regression, Decision Trees amongst others for a given dataset. As an example, given training and test dataset of spam and not-spam emails, classify the test dataset using Naive Bayes by training the algorithm. This example problem is a "spam classifier" algorithm. This requires students to perform the following actions: (i) Parse the training and test dataset, (ii) Split them into words, (iii) Remove 'stop-words' (commonly used words), (iv) Compute the word frequency, (v) Find the probability of occurrence of these words in spam/not-spam categories - the Naive Bayes algorithm (vi) classify the test dataset based on this learned data. The initial steps (steps (i) and (ii)) are the preprocessing of the data and is not part of the actual algorithm implementation. Our tool, JMLP (Java Machine Learning Project) provides an interface where such data preprocessing are abstracted and provides the words as a list or as a list of lists (Java lists) thereby allowing the students to only focus on the implementation and learning of the algorithm and not the language specific data loading and preprocessing.

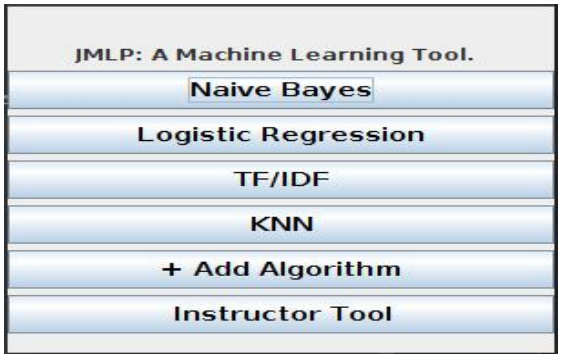

Fig 1: Home Window of JMLP

Our tool, JMLP, also abstracts other language-specific implementations, such as packages imports, implementation of Java classes and the main function. These are provided functionalities as part of our tool. Students find difficulties in debugging their logic and implementation since it mostly involves mathematics. To address this, we present a novel way where students can get immediate and relevant feedback on their implementations which will help them to understand their correctness, debug and modify their implementations. Our tool currently provides 4 algorithms for students to practice, which can be extended to any number of algorithms. Instructors can also add their course-specific algorithms which the students can practice, and also use them to evaluate the students' submissions.

Fig. 1 shows the interface of the main window of our tool, which consists of 4 algorithms currently; Naive Bayes, Logistic Regression, TF/IDF (Term Frequency/Inverse Document Frequency) of a document cluster and k-nearest neighbors (k-NN). Upon selecting one of the algorithms, 2 windows are displayed - the code editor window and the result window - as shown in Fig. 2. Students can edit their code on this code editor which has many of the features of a typical code editor - syntax highlighting, line numbers, machine braces highlighting, code folding, among others [3].

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

The students can implement algorithm using the data passed by the function parameter. Upon completing the implementation, the "Save to File" button is clicked, which saves the code in the code-editor to a file and this file is compiled in the background. If an error is found, it is displayed as a popup with formatted text. This way, the syntax errors can be corrected easily. The tool is implemented in Java and all the methods exposed to the students are Java methods. This is a valid choice as the course of Java programming is usually taught before the course of Machine Learning, and since Java is one of the most widely used programming languages, implementation of the solutions in Java by the students is a convenient choice. Also, given that the students do not have the need to implement the specifics of the Java programming language such as classes, encapsulation, object creation among others, it makes implementing the algorithm easier and allowing them to focus on the algorithm.

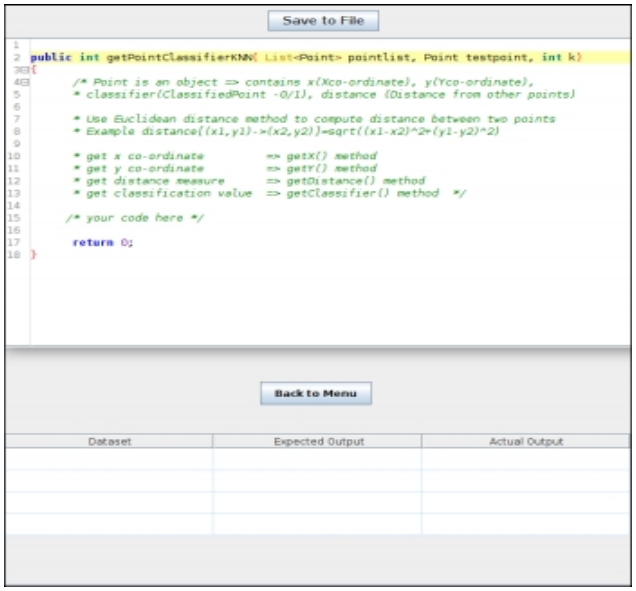

Fig 2: Code Editor and Result Window – a working example.

II. RELATED WORK

Weka is a tool consisting of various machine learning algorithms which can be applied on the user's data [1]. Since, it's initial release, it has been extended in various ways to support a wide variety of algorithms and nature of data. Scikit-learn [2] is a collection of python modules comprising of a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. Yet another open-source library is available – LIBLINEAR [4], which is mainly used for large scale linear classification and can be efficiently used for a large sparse dataset. Dlib-ml [5] is a tool which provides a rich environment for developing machine learning software in C++. Similar to this, yet another tool – Shark [6] also enables users to develop programs in C++.

All these existing tools are excellent tools which solve or provide a method (APIs) to use the machine learning algorithms on a user specific data in an efficient, reliable and easy-to-implement manner. But, for a beginner student, there is very little these tools have, to offer, to assist them in learning the algorithm. JMLP solves these problems by providing the platform to implement and test the solutions and get automatic feedback on their implementation with respect to valid reference implementations. It will help them to learn the concepts of machine learning in a better way with more knowledge and practice.

Also, this work incorporates the same underlying concepts from the first author's previous research - JFLAP enhanced (https://bit.ly/jflapenhanced) [7]. It provides a novel way for students to obtain feedback on their solutions. Subsequent research backs the efficacy and usefulness of such a tool. The research carried out a field experiment to understand the effectiveness and usefulness of automated feedbacks. The authors found out that both counterexample-based and the hint-based feedbacks were more effective than binary feedback in the students' learning process [8]. This motivated the authors of this paper, to build a similar tool in the domain of Machine Learning, as there is no such tool present in this domain, to the knowledge of the authors. The design principles and methodologies of this tool similar to JFLAP-enhanced, but some changes has been incorporated keeping the nature of the topic into consideration. Unlike the course of finite automata, this course requires training and testing data to work on, making the design and architecture different.

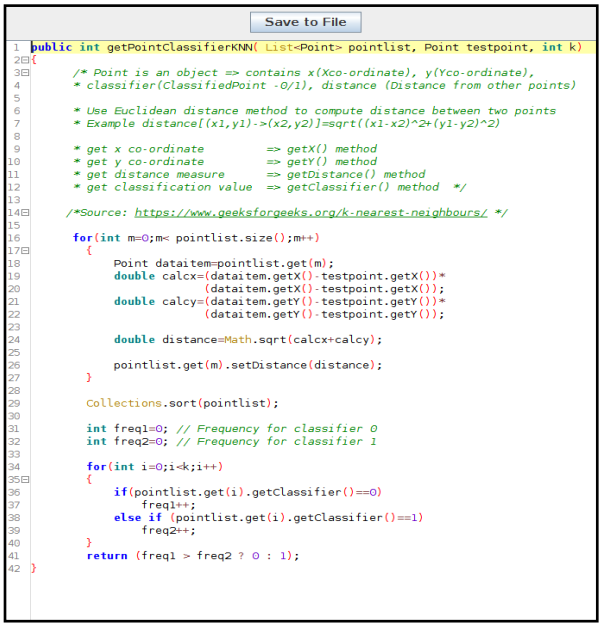

Fig 3: Reference implementation of k-NN algorithm.

III. STUDENT TOOL-AUTOMATED FEEDBACK

When students implement the algorithm, they get very little or no feedback about the correctness of their implementation. The students are made aware of their mistakes when the instructor releases the solutions, but then, the students will not be able to learn from these mistakes and make the necessary changes in their solutions and earn points or be evaluated for the same. One way for students to get immediate feedback is to develop test cases to test their solution, as per the practices of Test-Driven Development (TDD) but quite often, some crucial test cases or edge cases might be missed out, which on the contrary, since the instructors are aware of these cases, they will test these students solution with those test cases upon which these solutions will fail.

JMLP addresses these issues by providing a mechanism where students can get immediate and relevant feedback on

their implementations which can help them debug their solution. We currently provide 4 machine learning algorithms, to show the functionality and usage of this nature of tool, but it can be extended to any number of algorithms.

Students are given a Java function prototype with the corresponding data parameters, where the data is preprocessed and passed through the functional parameter. The students can use this pre-fetched data to implement the algorithm, and all the language specific constraints such as class, object creation, etc are handled by the tool.

For example, the k-NN algorithm, the students are given the function, **getPointClassifierKNN(List data, Point test)** where Point is a class and its object has the following values:

X coordinate - The 'x' coordinate of a point of consideration in a 2-dimensional plane.

Y coordinate - The 'y' coordinate of a point of consideration in a 2-dimensional plane.

distance - The Euclidean distance between two points (x1, y1) and (x2, y2).

classifier - The output of the k-NN algorithm - the class membership. Our reference implementation assumes two groups, viz. 0 and 1.

Fig. 3 shows the implementation of the k-NN algorithm used as a reference to test the students' implementation for example. It is to be noted that each algorithm has it's own and unique function signature (prototype) which is as required by the algorithm. Currently, our tool supports k-NN algorithm with points of 2-dimensions only, and we shall extend it to higher dimensions as well, in a future release.

The "data" is the training data which the student can directly use in their training implementation, and the "test" is the testing value which is used to test the algorithm. We provide the implementation of the preprocessing of data, loading it into memory using the appropriate data structure and remove noisy data.

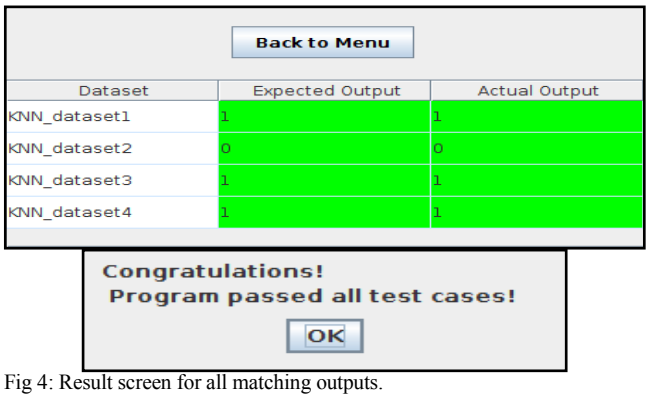

Fig 4: Result screen for all matching outputs.

After the student has implemented the algorithm, to verify it's correctness, he or she has to click on the "Save To File", upon which the content of the implemented code is encapsulated within a class, objects are created and then saved to a file. Then, it is compiled, and if no compilation error is found, it is run and the output is compared against the reference code and if all the test cases run successfully, then a prompt saying "Program Passed all test cases" is displayed, else, a comparison stating, "expected output" and "actual output" is compared and displayed, along with a hint of where the student might have gone wrong. This can help them debug and fix their code with ease, thereby improving the understanding of the algorithm better alongside being able to program better.

The tool displays the solutions mismatch with a red background while the matching ones with a green background, as shown in Fig. 4. This helps in identifying the correctness in an easy and visual way. Also, the popup will appear indicating all the wrong/mismatched outputs, along with some crucial feedbacks on where the mistake might have potentially occurred, to get this wrong output, as shown in Fig. 5.

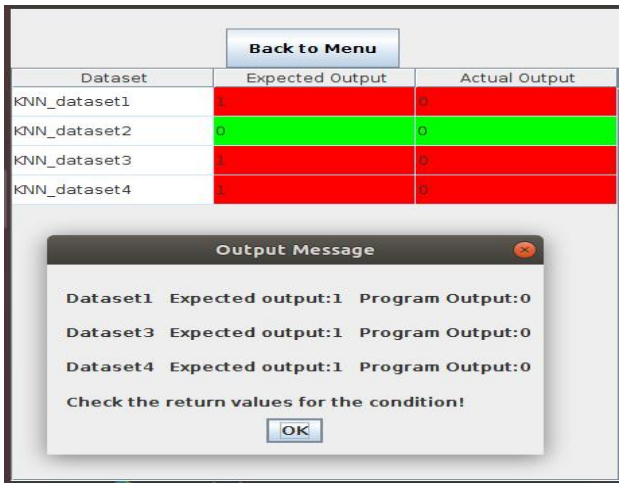

Fig 5: Result screen for wrong outputs, with feedback.

Sometimes, we observe that, some outputs may be displayed correct. These are false positives and we intend to detect them and provide more accurate feedbacks in the subsequent versions of our tool. In Fig.6, it can be noted that the student solution returned a negated value to the actual output. One possible reason for this is that the student has erroneously returned values in reverse. We currently work on the goal that **"If a solution is wrong or flawed, there exists at least one counter-example input which can produce such a wrong output."**

We note that the fact that there are no errors doesn't necessarily mean that the student solution is correct as it could be that the actual solution was very simple or had a specific pattern of output, for the given data, in which cases, there is little our tool can do, to detect the same, presently. But otherwise, for a vast majority of students, we believe that this tool is extremely useful as the feedback generated is automatic, and requires no human intervention, and also, let's you learn at your own pace. The feedbacks are immediate and personalized to the students' own approach, thereby making it usable to a vast range of audiences.

IV. INSTRUCTOR TOOL-STUDENT EVALUATION AND STATISTICS

JMLP also allows the instructors to automatically evaluate his or her students and get the statistics of the overall performance of the class. This saves the instructors both time and effort and provides them with a larger insight into the students' performances. The instructor can view the total number of students who have got the correct solution, an average score, the median score and the standard deviation of the class and also, we provide the statistics of

the number of correct and wrong outputs derived from the testing on a per student basis.

Fig. 6 shows the instructor tool, where they can select a directory of student submissions that they want to evaluate, and the algorithm that must be tested upon, and our tool provides the evaluated results. As shown in the figure, the statistics we currently provide include the total number of correct solutions, average score, median score, standard deviation, number of test cases each solution succeeded and the number failed. Upon selecting a student solution, it also displays which test cases it failed (the dataset it failed to produce similar results to the reference implementation) and the expected output and the student solution output. This provides the instructor a complete view of the coarse and fine-grained performance of their class. This is very helpful in large classrooms such as MOOCs where manual evaluation is near impossible, yet requires to provide near manual evaluation results with elaborate results.

The interface of the instructor tool is made intuitive and easy to use. This interface allows the course instructor to evaluate the student submissions against a reference correct solution. To do so, select the "Instructor tool" from the main window. This will pop-up a new window which allows the instructor to select the directory which contains all the student solutions and select the algorithm option from the drop-down menu that is to be evaluated. If it is not found in the drop-down menu, the instructor can add their own reference algorithm along with the dataset. Then this algorithm can be selected from the drop-down and can be used for testing or evaluating.

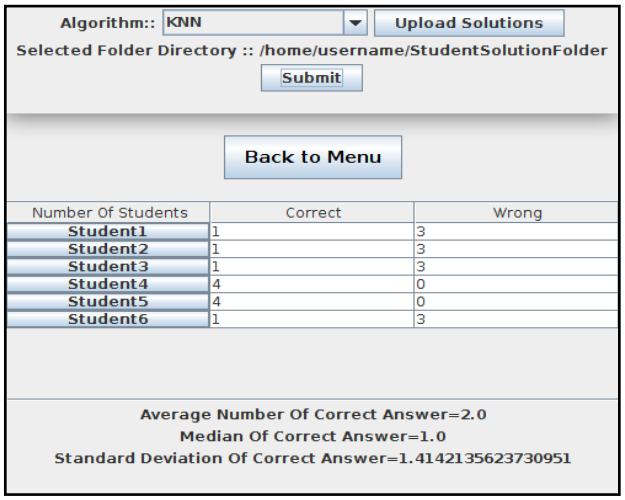

Fig 6: Instructor Tool window with student summary statistics

The benefits of such tools is two-fold - (1) The analysis and evaluation is immediate and (2) Provides near manual evaluation type of results which is beneficial for the instructor to understand the levels of the performance of his or her class, and in extension, for large classrooms like MOOCs, the nature and the diversity of the students who have taken up the course.

Since the dataset and the reference solution is the same for the student and the instructor tool, the effort of proving multiple reference solutions and datasets, to separately provide students with a reference implementation from which they can get feedbacks and another for the evaluation is unnecessary, in this case. Although, most of the times, instructors would like to provide students with a sample dataset for feedback and a different dataset for evaluation. We provide instructors a mechanism to only choose new datasets for an existing algorithm.

V. DISCUSSION AND FUTURE WORK

We have demonstrated how our tool can be useful to both instructors and students in the subject of Machine Learning. However, the measure of the short-term and long-term benefits it produces is yet to be measured. We are designing such a study which can be helpful to validate and measure the efficacy of the tool, but we believe that the tool will be of great use due to the success of two closely related tool and strategy: JFLAP[9] and the practices followed in the software debugging domain. Test-driven development is considered to be an effective strategy in software debugging and increasing the quality of code with over 45% fewer defects per 1,000 lines of code, according to a well cited study [10]. Based on these pieces of evidence, the authors advocate that this tool has its merits' and can be very useful in the area of Machine Learning. We are working on enhancing both the student and the instructor tool in the following ways:

The student tool will be enhanced by adding feedbacks based on the techniques of code comparison which involves comparing the generated bytecode with the reference solution's bytecode and providing feedback based on these differences. We believe that this nature of feedback would be more accurate and useful in terms of the logical errors created by the students in the code since these messages can be similar to compiler messages which are accurate and precise. Also, we are in the process of enhancing the tool to evaluate and provide a better user interface, in terms of the base code and the comments provided, which will enable the students to understand and use the tool better.

The instructor tool will be enhanced by providing better and more descriptive analytics which will include charts, graphs and figures along with some additional heuristics. We will also provide additional features such as common pitfalls and mistakes committed by the students thereby enabling instructors to fine-tune the topics of discussion in their classrooms accordingly. These will be made available in the subsequent versions and can be downloaded from the same link mentioned in the Section I of the paper.

Alongside these new features, future releases will allow integrations with popular tools such as Weka, making reference implementations to be directly connected to our tool from the already existing and validated algorithms from Weka. This reduces the effort from the instructor and gives students a wider range of algorithms to practice.

VI. ACKNOWLEDGEMENT

The authors thank Dr. Ram Rustagi, for all the guidance, support and the feedbacks given throughout the process of working on this paper.

REFERENCES

[1] Holmes, Geoffrey, Andrew Donkin, and Ian H. Witten. "Weka: A machine learning workbench." Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on. IEEE, 1994.

[2] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, *12*(Oct), pp.2825-2830.

[3] RSyntaxArea website: http://bobbylight.github.io/RSyntaxTextArea/

[4] Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R. and Lin, C.J., "LIBLINEAR: A library for large linear classification." *Journal of machine learning research* 9.Aug (2008): 1871-1874.

[5] King, Davis E. "Dlib-ml: A machine learning toolkit." *Journal of Machine Learning Research* 10.Jul (2009): 1755-1758.

[6] Igel, C., Heidrich-Meisner, V. and Glasmachers, T., 2008. Shark. *Journal of machine learning research*, *9*(Jun), pp.993-996.

[7] V. S. Shekhar, A. Agarwalla, A. Agarwal, N. B. and V. Kumar, "Enhancing JFLAP with automata construction problems and automated feedback," *2014 Seventh International Conference on Contemporary Computing (IC3)*, Noida, 2014, pp. 19-23.

[8] Loris D'antoni, Dileep Kini, Rajeev Alur, Sumit Gulwani, Mahesh Viswanathan, and Björn Hartmann. 2015. How Can Automatic Feedback Help Students Construct Automata?. ACM Trans. Comput.-Hum. Interact. 22, 2, Article 9 (March 2015), 24 pages.

[9] S. H. Rodger and T. W. Finley, "JFLAP – An Interactive Formal Languages and Automata Package", Jones and Bartlett, Sudbury, MA, 2006.

[10] S. H. Edwards, "Using Software Testing to Move Students from Trial-and-error to Reflection-in-action", SIGCSE Bulletin, vol. 36, no. 1, pp. 26-30, 2004.