Principles for using Machine Learning in the Assessment of Open Response Items: Programming Assessment as a Case Study

Varun Aggarwal
Aspiring Minds
varun@aspiringminds.in

Shashank Srikant
Aspiring Minds
shashank.srikant@aspiringminds.in

Vinay Shashidhar
Aspiring Minds
vinay.shashidhar@aspiringminds.in

Abstract

Questions demanding subjective (open) responses have been considered to be the most desirable assessment format in order to gauge candidate learning. Such questions allow candidates to express creatively and help evaluators to understand a candidate's thought process. The evaluation of such subjective responses, however, has traditionally required human expertise and is challenging to automate. On the other hand, automated assessments provide scalability, standardization and efficiency. Given the recent shift towards online learning and the massive scale of operations, there is a need to develop systems which can combine advantages of both, expert assessors and automated systems.

Drawing from attempts made by both, the machine learning community and educational phsychologists, we provide general principles on how any subjective evaluation problem can be cast in the framework of machine learning. These principles highlight the various choices and challenges one would need to consider while devising a machine learning based assessment system. We go on to demonstrate, as a case-study, how a system to assess computer programs has been successfully designed using the principles described.

1 Introduction

The assessment of open-response items, popularly known as subjective evaluation, provides a much more holistic and accurate assessment of a candidate as compared to closed response items (multiple-choice questions)[1, 2, 3]. Closed response items are wrought with many challenges. For certain tasks, they just don't work - consider assessing spoken English by asking a person to select the right pronunciation of a word rather than speaking it out. In principle, closed-response items (questions) ask the candidate to *choose* the right answer and not *find* the right answer, implicitly providing strong hints. A candidate who chooses the right answer may not have found that right answer had s/he not seen the options. Second, it provides binary assessment of right and wrong, where a part score cannot be awarded based on the approach followed by the candidate. As a result, one generally does not ask a problem requiring multiple solution steps in closed response items. Doing so, however, severely impedes their power to assess whether a candidate can take a series of right steps to solve a problem, which is usually the assessment objective in any field.

Among other approaches, people have successfully used machine learning for automatic subjective assessment. Some popular applications include the automatic evaluation of spoken English [4, 5],

essay grading [6] and our recent work in grading computer programs [7]. When compared to human assessment, a correlation in the range of 0.7 - 0.8 is observed for automatic essay evaluations, spoken English assessment and in programming assessment. Through these different works, we find the emergence of a set of general principles to guide the use of machine learning to automatically grade subjective responses. These principles provide a framework to think of how machine learning could be applied to automatically grade problems in a new domain (like evaluating responses which solve an electronic circuit), what design choices need to be considered and what challenges may occur.

In this paper, we espouse these principles through the case study of the automatic assessment of computer programs. The paper is organized as follows - Section 2 provides a framework for casting a subjective response grading problem into a learning problem, detailing the various steps involved. Section 3 describes a brief on how this framework was used to solve the computer program grading problem. Section 4 concludes the paper and discusses future work.

2 Open-response assessment as a machine learning problem

The machine learning/artificial intelligence community shall appreciate that such an assessment problem is a kind of an inversion of the classification/regression problem. In the latter, one tries to identify what an unknown object is, given it's quantitative parameters (features). In the assessment problem, we know what the object is supposed to be and now want to find out whether it belongs to the right category and if not, '*how close*' it is to the ideal. For instance, a classification problem could be to identify which drawings are those of a house, whereas the assessment problem would be that given a house had to be drawn, whether a house was drawn and how well it was drawn.

Building on this observation, we describe steps to convert a grading problem into a machine learning problem. We assume we have a question (or class of questions) whose open-responses have to be assessed using a statistically learned model.

a. Develop a scoring rubric: One must appreciate that the assessment objective is to determine 'how close' or 'how good' a solution is when compared to the correct solution. This is why one has to come up with what education psychologists call the scoring rubric [8]. A scoring rubric maps each score or a score range to the level of expertise shown by the person in solving a problem and helps in understanding what makes a solution closer to or farther from a good solution. For instance, drawing a house without a visible triangle placed on top of a visible square will be scored lowest followed by one which has these two entities drawn clumsily followed by those where they are placed correctly followed by the perfect house. The scoring rubric helps understand what *closeness* or *goodness* of a solution means for a particular question. One may note that the rubric may not necessarily be an ordered set of scores, but could also be a set of feedback categories for formative assessments.

Apart from this theoretical understanding, an analysis of candidate attempts[1] and an understanding of the human grading process helps form a scoring rubric. For instance, we analyze the human grading process for computer programs (see Section 3.1) to develop a rubric for their evaluation (Table 1). A scoring rubric is an important first step to decide upon in the process of automatic grading of responses.

b. Expert-graded solutions: For the purposes of learning, one needs to get a set of solutions for the given problems graded by experts according to the rubric. This provides the dependent variable in the machine learning formulation. Based on reliability considerations, more than one rater may be used and the ratings be combined through statistical means. The process to get trustworthy ratings has been discussed in detail in the past and could be referred to in [9, 10]. For human-intelligence tasks, crowd-sourcing may also be used, but only when they can be combined appropriately to mimic expert grading [11, 12].

c. Response format: The response format in most cases of subjective assessment is a free-flow format. For instance, students express their answers to a circuit analysis problem or an integration problem in free formats containing equations and explanations (and optionally diagrams too). On the other hand, there are problems whose solutions by default follow a fixed format with a strongly defined grammar, for instance in evaluation of computer programs. The fixed format provides two advantages: firstly, it facilitates easy derivation of meaningful features (implicit in the grammar

[1] Apart from qualitative analysis, one may also use unsupervised learning techniques to do so.

of the format) and secondly, allows the existence of execution engines (an interpreter/compiler in case of computer programs) to provide additional insight or features for the purposes of grading. Although being hard to deal with, features can be derived from open-format responses by parsing them using one or more expected grammars, as illustrated successfully for problems such as essay grading.

The examiner has the choice of asking the candidate to write solutions to questions in a given, fixed grammar. For instance, one could ask the test-taker to write the solution to a differentiation/circuit analysis problem in an equation format which, say, Mathematica can conveniently parse. Though this is advantageous from the point of view of the design of automated systems, it penalizes those students who are not comfortable with the given format and also restricts creativity. The format of responses is an important design consideration for the assessment problem and it's treatment using machine learning.

d. Features: Finding the right set of features for a given problem is most crucial for the success of a grading algorithm. The kind of features or their taxonomy should be directly inspired from the scoring rubric. One has to develop features which, in some combination, would be able to distinguish solutions graded at one level of the rubric from the other i.e. they need to correlate to the goodness of a solution, thereby suggesting how close a given solution is to a good solution. For instance, in measuring fluency, the scoring rubric distinguishes responses by the rate of speech, number of pauses, hesitations, etc. all of which are features derived by the alignment of speech on text [13]. In Section 3.3, we provide examples of features for the program grading problem and their linkages with the scoring rubric. One may note there that one of the features, the number of pre-compiled test-case passed by a sample program, is available because the response format is fixed in nature, with additional leverage being provided by the execution engine, namely the compiler. The definition of the scoring rubric and finding the right features inspired by this rubric are crucial elements in solving an open-response assessment problem.

e. Modeling: In principle, any regression or classification algorithm may be used to model the output grade based on the evaluated features. However, given that human-grading is expensive and sample sizes are limited and may have biases, there should be a preference to use either models informed by prior understanding of the problem space [14] or use interpretable models which can be verified by an expert [15].[2] On the other hand, availability of large amounts of data, which MOOCs can facilitate, can supersede the use of such approaches. In days to come, the field of developing problem independent features and reducing the need for a large number of expert-graded problems will be a major field of innovation.

A second important consideration is whether the model is for just a given question or a class of questions. For instance, in essay grading, models are generally trained for every prompt separately, although there has been some work in prompt independent grading as well [16]. To develop models which work for a class of questions is extremely useful and provides great scalability. A general way to do this is to transform features in a way using question-specific parameters, such that they are independent of the particular question's parameters in determining closeness to a 'good' solution. For instance, in the program grading problem, the number of test-cases passed is a feature which is independent of a given programming question. In essay grading, the word list whose frequencies are to be considered as features can be automatically derived from the word list of the prompt text, creating a generalized feature. Similarly, distances of solutions from good solution for each problem can provide general features.

A third consideration in modeling is to reduce the need for a high number of graded solutions. Here, research in one-class model building can come in handy. In the program grading example, we illustrate how by using only a set of automatically-identified good programs, we are able to build good grading models. This becomes more effective if the features capture the scoring rubric well and the distances in one's feature space directly correlate with distances in the grading space.

The above summarizes the various components of a machine-learning based automatic grading system. We now illustrate the above with our recent work on automatic assessment of computer programs.

[2] Interpretable models are very useful from another standpoint - they can help us understand the working of a human-grader's mind!

3 Computer Programming: A case-study

In this section, we illustrate how the framework described in the previous Section has been applied to the problem of grading computer programs. Sections 3.1 and 3.2 introduce a rubric to grade programs inspired by how human evaluators assess programs. Section 3.3 illustrates some informative features which were generated based on the described rubric. In Section 3.4, we discuss the machine learning framework used to solve the problem and also discuss some of our results.

3.1 Decoding the human evaluation process

We understand that to evaluate a given computer program, an evaluator looks for certain signature features. These features are in terms of specific control structures, keywords or data dependencies being present in the program. We observed that an evaluator first looks at the overall program to see if characteristic keywords specific to that problem are used at all. For e.g. in a problem which requires a pattern of digits to be printed, a characteristic keyword would be the '$print$' command. If such keywords are found, the next level of scrutiny takes place to ascertain whether the right control structures exist and if they do exist, whether these structures are organized in the right order, ensuring a correct flow in the logic. If the control structures used are seemingly germane to the problem at hand, finer aspects of the code are then looked at, such as terminating conditions of the control structures, the right dependencies between data-structures occurring in different parts of the code etc. In [7], we illustrate in detail how a simple problem is evaluated using such an approach.

3.2 Evaluation rubric

In Table 1, we present a problem-independent rubric to grade a program mimicking the human evaluation process. It objectively captures the program's state which, we hypothesize, links to the candidate's ability to develop an *algorithm* for a given problem. It can be used across problems and leaves scope to be more detailed and fine-grained.

Table 1: Rubric to grade computer programs

SCORE	INTERPRETATION
5	**Completely correct and efficient:** An efficient implementation of the problem using the right control structures, data-dependencies and consideration of nuances/corner conditions of the logic.
4	**Correct with silly errors:** Correct control structures and critical data-dependencies incorporated. Some silly mistakes fail the code to pass test-cases.
3	**Inconsistent logical structures:** Right control structures exist with few/partially correct data dependencies.
2	**Emerging basic structures:** Appropriate keywords and tokens present, showing some understanding of a part of the problem.
1	**Gibberish Code** Seemingly unrelated to the problem at hand.

3.3 Grammar of features

Motivated by the rubric's design, the feature space captures counts of expressions, control structures, data-dependencies and other properties that exist in a program [7]. The features contain information to be able to distinguish one level of the rubric from another. They are applicable to a program written in any programming language. Moreover, being generic, they are invariant of the underlying problem and to any particular implementation of a given problem.

One class of such features could include those obtained by counting the occurrences of various keywords and tokens appearing in the source code. These include keywords related to control structures

such as for, $while$, $break$, etc., operators defined by a language like '+','-','$*$', '%', etc., character constants used in the program like '0', '1', '2', '100', etc., external function calls made like $print()$, $count()$, etc. These counts are useful to see whether the right constructs even appear in a program (characterstics of Score 2, Table 1) irrespective of whether the control-flow and data-dependencies are right. Another feature class can include those capturing control flow counts. For instance, *LP_LP* can be interpreted as the number of times a nested loop (a for-in-a-for or a $while$-in-a-for etc.) appears in the program.

These features can be extracted from a combination of either the Abstract Syntax Tree of a program, it's Control Flow Graph, its Data Dependence Graph and its Program Dependence Graph. As long as the source code follows the grammar of the program strictly, these features are easily extractable. This generally is of great utility when a candidate is unable to complete his/her program in the specified time and submits a partial solution to the problem.

3.4 Machine learning approach

We cast the problem of automatic grading in the standard machine learning framework - where programs attempted for a given problem are rated by experts, following which a regression model is developed based on the proposed features. Given that a large number of features are generated, which include sparse features, a feature selection technique is used to provide better modeling generalization. One may also cluster features to reduce feature-dimensions.

We conducted experiments on two programming problems which were graded by domain experts. These problems, posed in the C programming language, were administered to over 550 senior-year undergraduate engineering students majoring in Computer Science or IT in India. They were delivered through *Automata*, an assessment product owned by Aspiring Minds[3], which provides an online compiler-based simulated environment. We used the test-case pass feature and the semantic features described previously to learn a model using Ridge Regression and SVMs, combined with different feature selection techniques. We also did preliminary investigations with one-class modeling techniques.

Table 2: Results on our feature-set

Problem	Cross-Val Error	Train r	Validation r	One-Class r
Prob-1	1.06	0.76	0.80	0.75
Prob-2	0.67	0.95	0.90	0.80

Table 2 reports the Pearson Correlation Coefficient (r) we obtained on using ridge-regression on our proposed features to predict the expert-rated grades for the two problems. In [7], we show that the features we propose add around 0.15 correlation points over the information available from just the testcase pass-fail status of the samples in the data-set. We also show that the class of features pertaining to control and data-dependencies between variables and expressions contribute around 0.3 correlation points over information obtained from just keyword and token counts. We are thus able to show the value-addition our rubric-motivated grammar of features is able to introduce to the problem of programming assessment.

The approach of using regression to build a model entails the requirement of a domain expert to be involved to rate the programs for each problem manually. In an attempt to do away or reduce this requirement, we also explore a novel way of posing this as a problem in one-class modeling. We do so by identifying high quality codes amongst the candidate submissions using an automatic technique which looks at the number of test cases passed, programming practices used and the complexity of the code (discussed in [7]). If the feature set indeed captures the functionality of the code and mimics the rubric, then a simple distance from these identified high-quality solutions in the feature space could provide the right grade (after some scaling). Whereas this distance approach mimics the neighborhood approaches prevalent in one-class classification (even though it is not classification), other approaches such as one-class SVMs, density estimation methods, etc. may also

[3] www.aspiringminds.in

be used. The column *One-Class* r in Table 2 reports the correlation between the predictions made using this one-class approach with the expert-rated grades.

4 Conclusion

The present work describes principles for guiding the development of assessment systems which use machine learning to evaluate open-response problems. We list out important steps to consider while designing such a system, namely, choosing a germane response format, creation of a robust rubric, capturing features which correspond to the developed rubric and choosing a machine learning model to predict a human expert's grades. In laying out these principles, we also highlight some essential factors and pitfalls one would need to consider during the system's design.

As a case study, we show how a system to automatically grade computer programs was designed using these principles. As a natural extension, these principles could also be used to build systems to assess other subjective problems such as evaluating electronic cicuits, high-school calculus problems or even in assessing applicants to blue-collared jobs. This work is timely with regard to the surge in the global outreach and participation in online education. We foresee a dominating role of reliable and robust ML-based assessment systems in the success of MOOCs and online education.

References

[1] Birenbaum, M. & Tatsuoka, K.K. (1987). *Open-ended versus multiple choice response formats - it does make a difference*, Applied Psychology Measurement, 11, 385-395

[2] C. P. M. van der Vleuten, G. R. Norman and E. de Graaff. (1991). *Pitfalls in the pursuit of objectivity: Issues of reliability*, Medical Education, 25, pp. 110-118

[3] Breland, H.M. (1983).*The direct assessment of writing skill: A measurement review* (College Board Report No. 83-6). New York: College Entrance Examination Board.

[4] *Svar, Aspiring Minds*, http://www.aspiringminds.in/talent-evaluation/spoken-english-SVAR.html

[5] Bernstein, J., Van Moere, A. & Cheng, J. (2010). *Validating automated speaking tests*. Language Testing, 27(3):355-377.

[6] Burstein, J., Braden-Harder, L., et al. (1998). *Computer analysis of essay content for automated score prediction: a prototype automated scoring system for GMAT analytical writing assessment essays* (ETS Research Report No. 98-15). Princeton, NJ: Educational Testing Service.

[7] Srikant, S., Aggarwal, V. (2013). *Automatic Grading of Computer Programs: A Machine Learning Approach*, International Conference on Machine Learning Applications (ICMLA), *to appear*.

[8] Moskal, B. M. (2000). *Scoring rubrics: what, when, and how?*. Practical Assessment, Research, & Evaluation, 7(3).

[9] Rudner, Lawrence M. (1992). *Reducing errors due to the use of judges*. Practical Assessment, Research & Evaluation, 3(3)

[10] Braun, H. I. (1988). *Understanding score reliability: Experiments in calibrating essay readers*. Journal of Educational Statistics, 13, 118

[11] Chklovski, T. and Gil, Y. 2005. *Towards Managing Knowledge Collection from Volunteer Contributors*. Proceedings of AAAI Spring Symposium on Knowledge Collection from Volunteer Contributors (KCVC05).

[12] R. Snow, B. O'Connor, D. Jurafsky, and A. Ng. (2008) *Cheap and fast - - but is it good?* In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 254-263. ACM Press.

[13] H. Franco, L. Neumeyer, Y. Kim and O. Ronen. (1997) *Automatic Pronunciation Scoring for Language Instruction*. Proc. Int'l. Conf. on Acoust., Speech and Signal Processing, pp. 1471-1474, Munich.

[14] P. Niyogi, F. Girosi and T. Poggio. (1998). *Incorporating prior information in machine learning by creating virtual examples*, Proc. IEEE 86(11), 2196-2209.

[15] J. Casillas, O. Cordn, F. Herrera, L. Magdalena (Eds.) (2003) *Interpretability Issues in Fuzzy Modeling*. Springer-Verlag.

[16] Higgins, D., Burstein, J., & Attali, Y. (2006). *Identifying Off-Topic Student Essays without Topic-Specific Training Data*. In J. Burstein and C. Leacock (eds), Special Issue of Natural Language Engineering on Educational Applications Using NLP.