Question Independent Grading using Machine Learning: The Case of Computer Program Grading

Gursimran Singh, Shashank Srikant, Varun Aggarwal
Aspiring Minds
{gursimran.singh, shashank.srikant, varun}@aspiringminds.com

ABSTRACT

Learning supervised models to grade open-ended responses is an expensive process. A model has to be trained for every prompt/question separately, which in turn requires graded samples. In automatic programming evaluation specifically, the focus of this work, this issue is amplified. The models have to be trained not only for every question but also for every language the question is offered in. Moreover, the availability and time taken by experts to create a labeled set of programs for each question is a major bottleneck in scaling such a system. We address this issue by presenting a method to grade computer programs which requires no manually assigned labeled samples for grading responses to a new, unseen question. We extend our previous work [25] wherein we introduced a grammar of features to learn question specific models. In this work, we propose a method to transform those features into a set of features that maintain their structural relation with the labels across questions. Using these features we learn one supervised model, across questions for a given language, which can then be applied to an ungraded response to an unseen question. We show that our method rivals the performance of both, question specific models and the consensus among human experts while substantially outperforming extant ways of evaluating codes. We demonstrate the system's value by deploying it to grade programs in a high stakes assessment. The learning from this work is transferable to other grading tasks such as math question grading and also provides a new variation to the supervised learning approach.

Keywords

Recruitment; Automatic grading; MOOC; Feature engineering; Supervised learning; One-class learning; Question independent learning

1. INTRODUCTION

The automatic grading of open-ended responses has become the subject of extensive research [9, 27, 10]. Machine learning and other computer science techniques are used to evaluate responses in a variety of domains such as essay writing [11, 24], computer programming [6, 25], spoken English [8, 22, 21] and open response math questions [16]. A number of innovations, in both academia and the industry, are focusing on open-response learning and assessment systems.[4, 2, 5, 1, 3]. MOOCs have a pressing demand for a scalable method of automatic grading of open-ended responses [18] and so does the industry.

Such assessments typically contain questions or prompts regarding a topic (hereafter referred to as *question*) to which test-takers must respond with written answers, for example, essay topics are prompts to test English, whereas programming problems are questions used to test software engineering skills. For each specific question, responses from different test-takers are used to build models that can predict the quality of new, unseen responses. For example, an essay describing a field trip to New York might include terms such as 'Statue of Liberty', 'Niagara', and 'Times Square' which would qualify the response to be rated highly; on the other hand, a *good essay* describing a visit to California might include such terms as 'Silicon Valley', 'Napa valley', or 'Golden Gate Bridge'. For each question, one has to identify such discriminating features and build models using labeled responses, making them specific to the subject matter of the question. The inherent design of such an approach impedes scaling these systems to newer content. Each new question requires a significant number of expert-graded responses to train models. The availability, cost, and time required by the subject matter experts (SMEs) govern the rate at which new questions could be added to the system.

We are interested in the case of automatic programming grading, where the difficulty is two-fold: first, each question needs to have its own predictive model; second, within each question, each specific programming language needs to have separate predictive models. With a programming evaluation platform supporting $5-10$ modern programming languages, scaling to new questions becomes very expensive. Additionally, there are constraints such as the investment of time and resources to train SMEs on the rubrics that they must follow in grading responses; the need to have at least two SMEs evaluating each response to ensure consistency in grading; and the scarcity of qualified SMEs.[1]

In this work, we address the problem of scalability in au-

KDD '16, August 13 - 17, 2016, San Francisco, CA, USA

ISBN 978-1-4503-4232-2/16/08...$15.00

DOI: http://dx.doi.org/10.1145/2939672.2939696

[1]One cannot make use of a crowdsourcing platform like Amazon Mechanical Turk to source SMEs because evaluating programming assignments are not human-intelligence tasks.

tomatic program grading. We present a method to grade computer programs which requires no human-graded samples for grading responses to a new, unseen question. For a given programming language, we transform question specific features to features that maintain their relation with the grade/label across questions. We design such a transformation by exploiting properties specific to the domain of computer programming. We call these transformed features as *expressive structurally invariant features* - their structural relation with the label remains invariant across questions. Using the structurally invariant features we learn a question independent supervised model, which can then be applied to (the automatically transformed features of) an ungraded response to an unseen question. In doing so, we extend our previous work where an expressive grammar of features was proposed and used to train question-specific supervised models. We also suggest how the learning from this work can be transferred to other problems of automatic grading. Specifically, this paper makes the following contributions:

- We present for the first time a machine learning approach to learn question independent models for grading computer programs (or for any domain, to our knowledge) which requires zero human-graded samples for a new question.
- We propose a new machine learning workflow to grade open-response problems, which includes automatic creation of structurally invariant features, normalization, a joint learning across questions with these features, and using them to predict labels for responses to an unseen question.
- Our approach produces a model for which the accuracy rivals the consensus among human experts and we demonstrate its utility in a real world setting to predict grades in high-stakes assessments.

The paper is organized as follows - §2 describes the *question independent* learning approach and related work. In §3, we present the details of our experiments and results. §4 discusses the deployment of our system in a high stakes assessment. §5 concludes the paper and discusses future work.

2. A QUESTION-INDEPENDENT LEARNING APPROACH

We describe in this section how question-independent features are engineered and used in a machine learning framework. We begin by laying out the notation, followed by a summary of the grammar of features proposed in [25]. This provides a context to the discussion in the subsequent sections.

2.1 Notation

For a given programming language, each programming question is referred to as a *question* and a code which attempts this question, as a *response*. The set of questions used in this study is denoted by Q_{all}. Each question $q \in Q_{all}$ has a set of responses $R^{(q)} \in R$ collected on it, where R is the set of responses across all questions. For any given response x, its feature vector is referred to as $\hat{f}_x$.[2] Each feature in feature vector $\hat{f}_x$ belongs to a category in set $\mathbf{C} = \{$B, BC, E, EC, ED, EDC$\}$ (see §2.2 for details). For a *question* $q \in Q_{all}$ having n responses and m unique features, we define the feature matrix $\mathbf{F}^{(q)}_{n\times m}$ as

$$\mathbf{F}^{(q)}_{n\times m} = \begin{bmatrix} \hat{f}_1 \\ \vdots \\ \hat{f}_n \end{bmatrix} \text{ where } \hat{f}_x \in \mathbb{N}^m \quad (1)$$

where $\hat{f}_x$ is the feature vector of a particular response x to question q.

We define a *good set* for each question (see §2.3 for details) $q \in Q_{all}$ containing k *good responses* as $\mathbf{R}^{(q)}_g$ and the feature matrix of the k *good responses* $\mathbf{G}^{(q)}_{k\times m}$, as

$$\mathbf{G}^{(q)}_{k\times m} = \begin{bmatrix} \hat{g_1} \\ \vdots \\ \hat{g_k} \end{bmatrix} \text{ where } \hat{g}_y \in \mathbb{N}^m \quad (2)$$

where $\hat{g_y}$ is the feature vector of a particular *good response* y to question q.

2.2 Feature Grammar

The grammar proposed in [25] generates features that capture the semantic relationships present in a program. The counts and relations of three important properties of a program are extracted - the tokens used in the program, the expressions and the data dependencies.

a) Basic (B) : Examples include counts of all keywords, tokens, operators etc. such as the number of times a '+' operator appears or loop tokens such as '*for*' and '*while*' appear.

b) Expressions (E): Expressions such as $y = x\%2$ are abstracted to a notation such as v:2::op:%::c:'2', which denotes an expression having two variables, one modulus operator and the constant '2'. The number of occurrences of such abstract expressions are counted.

c) Expression Dependency (ED): A data dependency is captured when the variable in one expression is used in another expression. For example, the expression $x < y$, which contains a relational operator ($<$) and two variables, is dependent on the expression $y{+}{+}$, which contains a post-increment operator ($+{+}$) and one variable. This is denoted using the notation v:1::op:$+{+}\leftarrow$ v:2::op:relation. The occurrences of each dependency matching such abstractions is counted. This is repeated for each unique pair of dependencies that appears in a response.

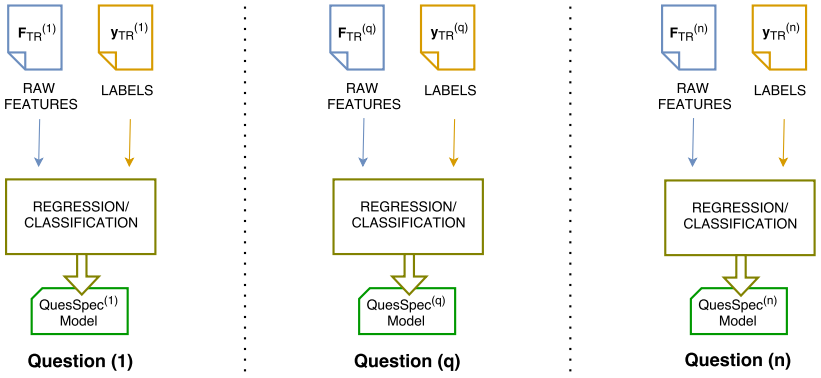
d) Control Context (BC, EC, EDC): Separate counts are maintained for each of the three properties described above according to the control-context (loops and conditional statements) in which they appear. For example, an expression whose abstract notation matches v:2::op:%::c:'2' is counted separately if it appears within an *if* statement as against in a loop like a *for* or a *while* as against in an *if* statement within a *for*.

The grammar of features described above is combinatorial in nature; a $10-15$ line program on an average generates 2000 features. A detailed discussion of the features is available in §2.3 of [25].

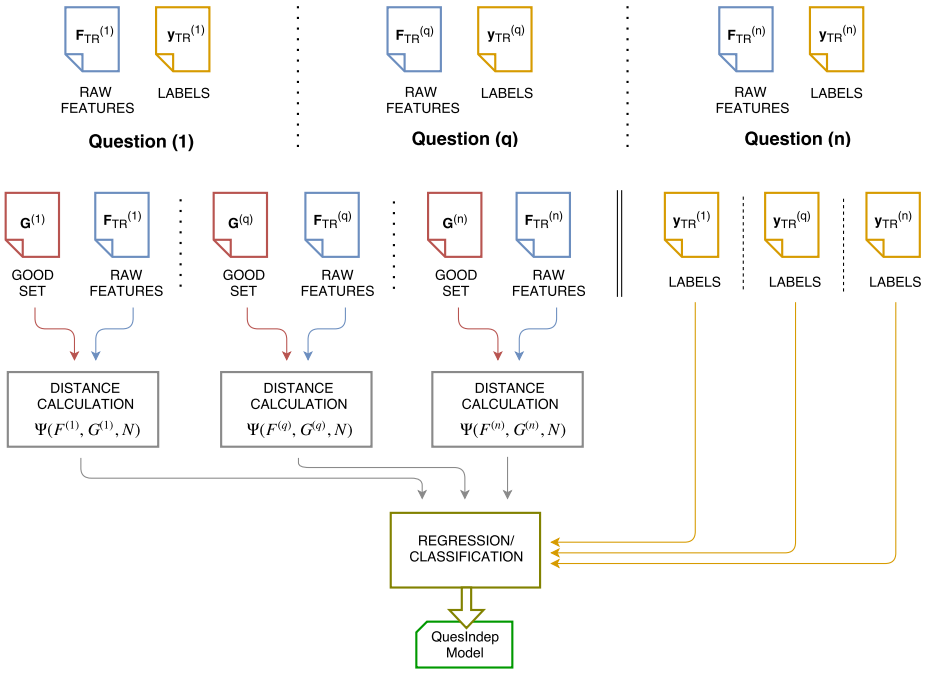
2.3 Method

The features introduced in [25] predict responses' grades accurately. However, the system requires to build a model

[2]All vectors represented by a cap ($\hat{f}_x$) are row vectors



(a) Question dependent model training



(b) Question independent model training

Figure 1: Question independent model learning

Fig (a) shows how question specific models were learnt in [25]. Each question required a set of grades ($y_{TR}^{(q)}$) corresponding to the feature matrix ($F_{TR}^{(q)}$) to train the models. A model was trained for every question separately. These models predicted grades of unseen responses for their respective questions. Fig (b) shows a modified workflow where only one model is learnt in the training phase. While training, each question's feature matrix first goes through a transformation Ψ before being augmented with the transformed matrices from other questions. This augmented input is used to learn the augmented grades of all responses used in the training phase.

for each question separately using labeled responses (see Fig 1a). Since getting responses labeled is a cost intensive process, we explore the design of a single model that predicts a response's label irrespective of the question it solves. Designing such a model does not fit seamlessly into traditional machine learning approaches since the features which discriminate 'good' responses from 'bad' ones change drastically across questions. For instance, we expect a nested loop in a 'good' response for bubble sort while one loop and other discriminating expressions is expected in a 'good' response to reverse a string. We wish to design features which structurally maintain the same relation with the grades irrespective of the question. One such relationship is an increase in the value of a feature, irrespective of the question, signaling a better grade.[3] We could then learn one model across responses from multiple questions in this invariant feature space and use it to predict grades of responses to unseen questions.

[3]A monotonic relation between a feature and a label is used for a simple illustration; other relations, like a quadratic form, are also acceptable in principle as long as they maintain this form across questions.

Mathematically expressed, we would want to design a transformation Ψ which would transform a *question specific* feature matrix $\mathbf{F}^{(q)}_{n\times m}$, for a given question q, into a structure invariant feature matrix $\mathbf{D}^{(q)}$. Although the transformation Ψ takes in *question specific* parameters, it must be automatic and not require any manually assigned labeled data for a response to an unseen question. The new invariant feature matrix $\mathbf{D}^{(q)}$ can then be augmented across questions and be used as one consolidated input to learn a *question independent* model **QuesInd** (see Fig 1b). The new *question independent* model can then be used to predict the grades of a new question without any human-graded samples.

We describe how we design the transformation Ψ which transforms the question specific features into a set of sufficiently expressive structurally invariant features.

a) Automatic identification of a *good set*: The *good set*, which we define as a subset of the responses that solve a question correctly, is vital in transforming *question specific* features into structurally invariant features (input to Ψ in Fig 1b). Although constructed for every question separately, creating this set does not require manually assigned labeled responses for a question. We posit that a carefully designed test-suite for a question automatically identifies functionally correct responses, which in turn can comprise the *good set*. This is not to say that responses failing a test suite cannot be good - in our approach, it is sufficient to have a subset of such *good responses* (corresponding to labels 4 or 5 of the rubric [25]).[4] We require a sufficient number of such responses so that they capture the variation in possible correct approaches to a question. We exploit the ability to automatically identify such a good set for a question to develop structurally invariant features. We now describe how we develop *structurally invariant* features from such an automatically identified *good set*.

b) Distance from a *good response*: For the sake of simplicity, let us assume the existence of only one possible *good response* to a question. In such a case, an L_1 distance (in the space of m features) between a given response and the *good response* will be a *structurally invariant* feature. A response at a larger distance from the *good response* will have a larger proportion of *different* 'keywords', 'control structures', 'data dependencies' etc. as compared to a response with a lower distance. Thus, responses with higher distances should probably have lower grades as compared to ones with lower distances, irrespective of the question the response and the *good set* belongs to. Such an L_1 distance from an automatically identified *good response* is hypothesized to relate to the grade in a *structurally invariant* way across questions.

We modify the L_1 distance to a one-sided distance ζ, defined as

$$\zeta(\hat{f}_x, \hat{g}_y) = \sum_{i=1}^{m} \beta(g_{yi} - f_{xi})$$
$$\beta(a) = \begin{cases} a & a \geq 0 \\ 0 & otherwise \end{cases} \tag{3}$$

where $\hat{f}_x$ is the feature vector of the response x and $\hat{g}_y$ feature vector of a *good response* y.

We do this guided by the intuition that programming constructs like keywords, control structures etc. not present in the feature vector ($\hat{f}_x$) of response x but present in the feature vector ($\hat{g}_y$) of *good response* y signals a deviation from how response x ought to be written. On the other hand, any excess of such features present in response x might still signal the approach used in *good response* y[5].

This still leaves us with two problems. First, there are usually not one but multiple ways to correctly implement a question. The distance ζ defined above can signal nearness to a specific *good response*. We would want a metric which could signal nearness to one among many such *good responses*. Second, the process of calculating ζ transforms the m dimensional feature space, consisting many interesting and useful features, to just one monolithic distance. We would want to continue having expressive features to learn models which generalize well. We address these problems below.

Algorithm 1 Distance from a *good set*

INPUT:
$\hat{f}_x$: Feature vector of a response x
$G^{(q)}$: *Good set* feature matrix of question q
N: N=1 to normalize distances

OUTPUT:
Distance of a response x from the *good set* of a question q

Initialize:
  $\hat{d}_{x\_all} \leftarrow null$ ▷ Dists to all *good responses*
function goodSetDist($\hat{f}_x$, $G^{(q)}$, N)
  for $y \leftarrow 1$ to k **do** ▷ Loop over all *good responses*
    $d_{xy} \leftarrow \zeta(\hat{f}_x, \hat{g}_y)$ ▷ See Eq (3)
    if N==1 **then**
      $(d_{xy}) \leftarrow \Omega(d_{xy}, \hat{g}_y)$ ▷ See Eq (4)
    $\hat{d}_{x\_all} \leftarrow \hat{d}_{x\_all} \cup (d_{xy})$
  $d_x = \textbf{\textit{meanMin25}}(_{x\_all})$ ▷ Mean of min 25
return d_x ▷ Distance from *good set*

c) Distance from a *good set*: A question typically contains multiple *good responses* in its *good set*. A given response has a distance ζ from each of these *good responses*, which we aim to aggregate meaningfully. Such an aggregate could be calculated in many ways, the mean and the minimum of the distances from the *good set*'s responses being two examples. It was shown in [25] that the mean of the 25% minimum distances correlated moderately with the output and outperformed the mean and the minimum. It discussed

[4]There could be cases where a response attempt to trick the system by hard-coding the exact test cases being evaluated. In both, our system and the system proposed in [25], this was handled by keeping a good number of test cases in the test-suite hidden from the respondent, making it hard to guess which specific cases a response would be evaluated on.

[5]We also conducted experiments, not reported for brevity, whose results validate the intuition empirically.

that the mean would dilute the distance metric from matching the closest correct response and the minimum would be too sensitive to outlier responses in the *good set*. We use this mean of the 25% minimum distances across *good responses* as the aggregate measure. Algorithm 1 describes the distance of a response x belonging to a question q from its *good set* $\mathbf{G}^{(q)}$.

d) Expressive distance scores: In our discussions thus far, we obtain a single monolithic distance from Ψ as a feature, for a response, to signal 'goodness'. This distance is aggregated across all features ($\mathbf{F^{(q)}}$) and across the different responses in the *good set*. This is in contrast to the question specific approach, wherein we had the advantage of having several varied features to learn a model. We address this by aggregating the distance across each of the six feature categories one at a time (see §2.2 for a description of the categories). We rewrite the features matrices for the responses and the *good set* as -

$$\mathbf{F}^{(q)} = \begin{bmatrix} \mathbf{F}_B^{(q)} & \mathbf{F}_{BC}^{(q)} & \mathbf{F}_E^{(q)} & \mathbf{F}_{EC}^{(q)} & \mathbf{F}_{ED}^{(q)} & \mathbf{F}_{EDC}^{(q)} \end{bmatrix}$$
$$\mathbf{G}^{(q)} = \begin{bmatrix} \mathbf{G}_B^{(q)} & \mathbf{G}_{BC}^{(q)} & \mathbf{G}_E^{(q)} & \mathbf{G}_{EC}^{(q)} & \mathbf{G}_{ED}^{(q)} & \mathbf{G}_{EDC}^{(q)} \end{bmatrix}$$

We then calculate Ψ with each of the six input pairs ($\mathbf{F}_B^{(q)}$, $\mathbf{G}_B^{(q)}$), ($\mathbf{F}_{BC}^{(q)}$, $\mathbf{G}_{BC}^{(q)}$) and so on, resulting in six distance scores for a response. This results in one distance feature corresponding to only *Basic* features like counts and tokens, one corresponding to *Expression* features etc. We hypothesize these categories to have different relations (weights) with the grade despite each of them being structurally invariant across questions. For instance, the *Basic* feature may relate more to gross differences in the responses and thus help signaling large jumps across rubric levels while the distance in the *Expression Dependency* may help quantify fine differences between responses, differentiating at the upper end of the rubric.

e) Transformation, Ψ: The distance score calculated for any response x of question q from the *good set* of q is the transformed feature value which is used to learn question independent models. For a question with n responses, the transformation Ψ converts the feature matrix $\mathbf{F}_{n\times m}^{(q)}$ into a vector of distances $\mathbf{D}_{n\times 1}^{(q)} = \begin{bmatrix} d_1 \dots d_n \end{bmatrix}^T$ where each d_i is the distance of response x from the *good set* $\mathbf{G}^{(q)}$.

f) Normalization, Ω: We describe how a distance score is calculated for a response by aggregating its distance from all responses in a *good set* of a question. Although the aggregation ensures that the general relationship between the distances and grades is maintained across questions, the scale of the distances could vary with questions. Specifically, Equation (3) scales up by the number of features which varies across questions. For instance, the set of unique features in responses implementing bubble sort (∼11 lines of code, $O(N^2)$) would be much more than to swap two numbers (∼4 lines of code, $O(1)$). As a consequence, the magnitude of distance for a 'nearer' response may turn out to be much larger for bubble sort when compared to swapping two numbers. This creates an issue when we augment these differently scaled distances for a joint learning task. A simple way to circumvent this is to normalize the aggregate distance by the total number of features in the *good response* y that it is compared to. We such define a normalization Ω as

$$\Omega(d_{xy}, \hat{g}_y) = \frac{d_{xy}}{\mu_y} \tag{4}$$

where d_{xy} is the distance of response x from *good response* y and $\mu_y = \sum_{i=1}^{m} g_{yi}$ is the sum of features in the *good response* y.

Another normalization strategy could be to normalize by the average distance among responses in the *good set*. One may see this as using distributional properties of an unlabeled sample. We see in the results section §3.3 that some of these normalizations are useful. In summary, by automatically identifying *good responses*, we are able to create six *structurally invariant* features, with a hypothesis that they relate differently and could add incremental value over each other to predict the grade.[6]

2.4 Related Work

Recent work in automated programming grading has focused more on providing feedback than a grade. [23] and [20] propose a program synthesis approach and a static analysis approach respectively to provide feedback for erring programs. These approaches find if small modifications could be made to the program to enable it to pass their respective test cases. They provide a list of such modifications as feedback to students. The systems seemingly perform as expected only on very small and non-complex pieces of code (less than $10-15$ lines implementing simple algorithms) and work only for responses with small deviations from a specified good response. Moreover, they do not focus on evaluation against multiple correct responses. [14] and [15] each propose a system which uses unsupervised learning to cluster together semantically similar codes. An SME provides a grade to any one representative code from each cluster. This grade is then propagated to the rest of the responses in that cluster. These systems have not been designed to automatically grade responses at scale in real time and instead focus on reducing the workload of an SME who is involved in the feedback generation process. This is very different from the business requirements of high-stakes automatic grading of computer programs. In comparison to these systems, ours is the first approach to automatically generate a grade for a program written for a given question without having any manually assigned labeled samples for it. Also, we use supervised learning techniques and derive features from a more expressive feature grammar.

Among machine learning techniques, the literature of domain adaptation [13], which constitutes a sub problem in the field of transfer learning [7, 26], addresses a similar problem as ours. In domain adaptation, a well established model built on data drawn from a *source* distribution is used to learn a model on data drawn from a *target* distribution. The task to be accomplished (e.g. tagging natural language) remains the same across the domains while some distributional properties of the features change. In our problem, unlike in domain adaptation, the learning task changes significantly[7]

[6]In principle, we could treat every original feature separately, convert them into a distance, align them across questions and use them to learn a question independent model. This has issues of high sparsity and learning complications because many features are unique to a question.

[7]Even though the task sounds the same, i.e. to grade pro-

from one question to another. The feature space and the discriminating features drastically change across questions. From the standpoint of domain adaptation, it is hard to say what is the commonality between the source and the target models for different programming questions - there doesn't seem to be any connection between the features or the structure of the models for, say, a bubble sort and a tree traversal. However, if one is able to define structurally invariant features as we do, some learnings from the domain adaptation literature may be useful.

3. EXPERIMENTS

We designed our experiments to address the following questions

- Is the **QuesIndep** model a good predictor of programming performance when compared to human experts? Does it perform well enough to be used in high stakes computer programming assessments?
- How well does the **QuesIndep** perform against **QuesSpec** models and the test case metric? The test case metric provides a baseline while the **QuesSpec** models, with their large feature sets, provide an upper bound on the accuracy of predictions.
- Do normalizing the structurally invariant features help build better models?

To answer these, we analyzed 19 programming questions[8] graded by experts trained on our rubric. The responses to these questions were, on an average, $30 - 35$ lines long and required a variety of data-flow dependencies and control structures to be implemented. We experimented with both, linear and non-linear machine learning techniques. For each question, we learned a specific model and compared its performance on a question-independent model learned on a subset of the questions in the data set. We analyze the performance of the models aggregated over all the questions as well as separately for each question. We now discuss the details of the experiments.

3.1 Data

The experiments were run on a set of programming questions hosted on *Automata*, our automated programming evaluation platform [25]. Respondents, who were college seniors majoring in computer science, took a 90 minute assessment in a proctored environment wherein they attempted two programming questions in a language of their choice. For these questions, the respondents had a choice of C, C++ and Java. A total of 19 questions were used in the study (see Table 1), on which we had an average of 285 responses per question and 5434 responses in all. The topics covered by these questions spanned iterative/ recursive algorithms, trees and graphs and other algorithms like the shortest job first etc.[9]

grams, it actually is different. For instance, the task is to determine whether a response implements bubble sort as against finding whether a number is prime.

[8]Models were built separately on data collected in three languages - C, C++ and Java. Wherever question independent models are mentioned, we refer to the average performance of the models across the three languages.

[9]Due to a paucity of space, we do not list here the details of the question statements.

Table 1: Details of our dataset

Question Name	#Codes	#Good Set	#Features
Qused_tr			
isSameReflection	126	69	2874
waitingTimeSJF	307	31	4513
countCacheMiss	455	93	3682
isTree	420	71	2741
patternPrint	603	211	2945
grayCheck	610	134	1793
transposeMultMatrix	507	183	2728
eliminateVowelString	515	156	2424
Total	**3543**	**948**	
Qunseen_tr			
cellCompete	215	70	2801
insertSortedList	101	22	1154
isSubTree	104	39	1315
minTreePath	189	76	1255
isPath	71	27	4766
lruCountMiss	244	66	4416
generalizedGCD	248	72	3163
distinctElementCount	256	91	1656
rotatePictureMethod	209	83	2085
reverseLinkedList	41	19	1162
balancedParentheses	213	96	1649
Total	**1891**	**661**	

The number of overall responses and the responses in the *good set* for each question is listed. Questions are divided into two sets - Q_{used_tr} and Q_{unseen_tr} based on how they are used in training the models (see §3.2.1)

Three professional software engineers with 3-5 years' experience each, who were also seasoned competitive programmers, shared the task of grading the responses. Each response was graded by two experts. The experts followed the rubric defined in [25] to grade codes on a scale of 1-5. Before beginning the grading exercise, they underwent a one-week workshop wherein they learned how to interpret the rubric and participated in mock grading exercises. The correlation between the grades of any two experts on an average was 0.81 across the questions in the data set.

We note here that grades are required only once for responses to only a subset of these questions. A model built on such responses then predicts grades of responses to any number of unseen questions without manually assigned labeled responses.

3.2 Models

We developed machine learning models on the responses collected on the questions listed in Table 1. We used linear regression, linear regression with L_1 regularization (LASSO), linear regression with L_2 regularization (Ridge regression), decision trees, random forests and SVMs. For LASSO[17] ($\alpha = 1$) we varied λ from 0 to 4. For ridge regression, we varied λ from 0 to 100. For random forests[19], we varied the number of estimators from 15 to 100. For SVMs[19, 12], we tested three kernels - linear, polynomial and RBF. We varied the penalty factor C from 0.125 to 128, and parameters γ from 0.125 to 128 and ϵ from 0.5 to 16. In all these techniques, the model which gave the best cross-validation correlation was selected. For the question specific models, a feature selection step was followed by a 3-fold cross val-

Table 2: Results of LASSO on the test set - Mean values across questions

Metric	Question Set	#Questions	QuesSpec-All	QuesSpec	QuesIndep-N1	QuesIndep-N0	Baseline-TC
Correl	Q_{all}	19	0.84	0.81	0.80	0.76	0.65
Bias	Q_{all}	19	0.14	0.15	0.24	0.28	0.35
MAE	Q_{all}	19	0.41	0.46	0.58	0.66	0.85
Correl	Q_{unseen_tr}	11	0.85	0.81	0.80	0.76	0.65
Bias	Q_{unseen_tr}	11	0.14	0.20	0.27	0.34	0.31
MAE	Q_{unseen_tr}	11	0.43	0.46	0.62	0.70	0.84

idation. The best cross-validation correlation was used for selecting the final model. Similar to what was seen in [25], linear models worked the best among all these techniques, indicating linearity in the inherent structure of this problem space. We report results only for LASSO in this work which outperformed all other techniques. We trained the following models to answer the questions enumerated at the beginning of the section:

- **QuesSpec-All**: We train separate models for each question as described in [25]. The full set of features (with feature selection) were used as inputs. This provides the upper bound on accuracy for our techniques.
- **QuesSpec**: Separate models were trained for every question using the six *expressive distance scores* (see §2.3d) as features. This provides an upper bound on what best we could do with the six categorized features. Our question independent model cannot do better than this (other than in cases where models overtrain). If this model does not perform well, then the technique we introduce in this work cannot be expected to do well either. On the other hand, it is useful to find how the question independent model does in comparison with these.
- **QuesIndep**: Our question independent model was trained across a subset of questions (discussed in §3.2.1) using six of the *expressive distance scores* as features. We train separate models in this category with and without applying the normalization (Ω), discussed in §2.3f. We denote these models by the names **QuesIndep-N1** when normalization is applied and **QuesIndep-N0** when it is not.
- **Baseline-TC**: The baseline for all our experiments is a model trained on just the test-case scores across questions. This mimics a real world scenario where we could have predicted the 'goodness' of a response by its performance on its test-suite. Given that the performance on a test-suite is a metric whose meaning does not change across questions, it too is a *structurally invariant* feature whose values can be used across questions to learn a model.

In all the above models, the percentage of test-cases passed was added as a feature.

3.2.1 Train-Test Split

To learn the **QuesSpecAll** model, we split the responses to each question into a $67-33$ train-test set. The selection of the train-test set was nuanced for the **QuestIndep** models. In addition to testing the model on responses belonging to questions already present in the train set, we ensured that there were responses in the test set from unseen questions - those whose responses were not used to train the models at all. The performance on such an unseen set would then demonstrate how well the **QuesIndep** models generalized to questions where we did not have any human-graded samples, the use case expected in a real-world scenario. We also ensured that the responses in the *good set* for each question were not a part of their respective test-sets. We denote those questions whose responses are used to train the question independent models as Q_{used_tr}. Questions whose responses are not used in training the models are denoted as Q_{unseen_tr}. Q_{all} denotes the set of all the questions. In our data-set, we randomly selected 8 questions as part of Q_{used_tr} and the remaining 11 as Q_{unseen_tr}. We trained the **QuesIndep** models on 67% of the responses belonging to Q_{used_tr} (2384 in total) and tested them on the remaining 33% responses of Q_{used_tr} and additionally on all responses of the Q_{unseen_tr} questions (3050 in total).

3.2.2 Evaluation Metric

We use the Pearson correlation coefficient (r) to quantify and measure the similarity between the predicted grades and the experts' grades. While describing the **QuesIndep** model building process (see §2.3), we suggested how the distances calculated from the *good set* could have been on different scales for different questions. For some questions, this could cause the question independent model to scale and/or displace the distribution of grades linearly as compared to their true distribution. As a consequence, the model may still correlate highly with the grade. We hence consider the bias[10] ($\sum \frac{y_{pred}-y}{n}$) and MAE ($\sum \frac{|y_{pred}-y|}{n}$) as evaluation metrics in addition to the correlation coefficient.

3.3 Discussion

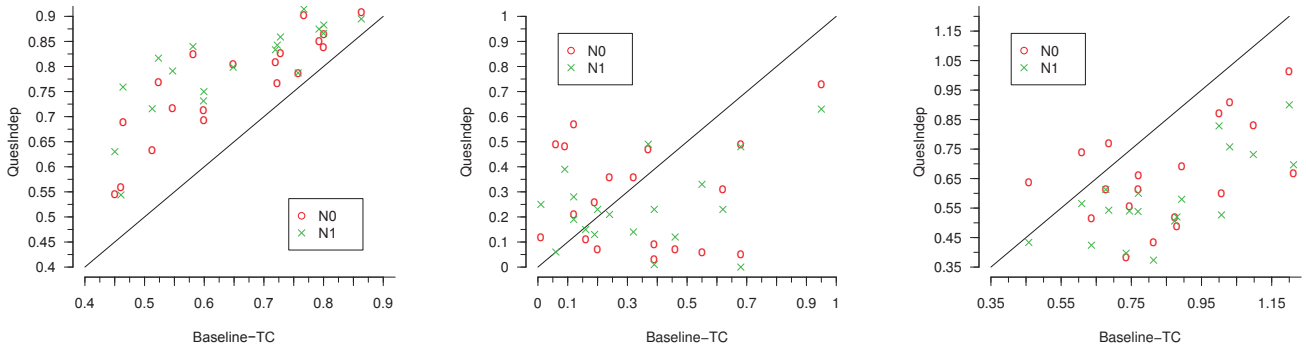
We contrast the performance of the **QuesIndep** and **QuesSpec** models on the test set (Table 2). To maintain brevity, we tabulate the mean of the models' performance on each question. We compare the performance of the models on two sets of questions - Q_{all} and Q_{unseen_tr} (see §3.2.1). The baseline results for the Q_{unseen_tr} are calculated by learning a model on the test case scores of only the Q_{used_tr} set and testing on the Q_{unseen_tr} set. The **QuesSpec** results on the Q_{unseen_tr} set correspond to the models learned on each of the Q_{unseen_tr} questions.

We first discuss the aggregate results across questions. We find that the question independent model with normalization (**QuesIndep-N1**) has an r of 0.80 with expert grades on both sets of questions - Q_{all} and Q_{unseen_tr} with a bias of 0.24 and 0.27 on the two sets respectively. This rivals the

[10]The bias in this work does not refer to the bias of learning models. It refers to the observational error.

Table 3: Results of LASSO on the test set - Question-wise results

		QuesIndep-N1			**Baseline-TC**		
	Question Name	Correl	Bias	MAE	Correl	Bias	MAE
Q_{used_tr} ($N = 1159$)	isSameReflection	0.54	0.63	0.73	0.46	0.95	1.10
	waitingTimeSJF	0.80	0.12	0.52	0.65	0.46	0.88
	countCacheMiss	0.82	0.01	0.53	0.52	0.39	1.01
	isTree	0.83	0.23	0.54	0.72	**0.20**	0.74
	patternPrint	0.88	0.14	0.42	0.80	0.32	0.64
	grayCheck	0.84	0.25	0.60	0.72	**0.01**	0.77
	transposeMultMatrix	0.87	0.00	0.37	0.79	0.68	0.81
	eliminateVowelString	0.79	0.15	0.58	0.55	0.16	0.89
Q_{unseen_tr} ($N = 1891$)	cellCompete	0.73	0.06	0.54	0.60	0.06	0.69
	insertSortedList	0.76	0.33	0.70	0.46	0.55	1.21
	isSubTree	0.75	0.21	0.76	0.60	0.24	1.03
	minTreePath	0.79	0.13	0.61	0.76	0.19	0.68
	isPath	0.86	0.39	0.57	0.73	**0.09**	0.61
	lruCountMiss	0.84	0.23	0.51	0.58	0.39	0.87
	generalizedGCD	0.72	0.48	0.90	0.51	0.68	1.20
	distinctElementCount	0.89	0.19	0.43	0.86	**0.12**	0.46
	rotatePictureMethod	0.91	0.28	0.40	0.77	**0.12**	0.74
	reverseLinkedList	0.87	0.23	0.54	0.80	0.62	0.77
	balancedParentheses	0.63	0.49	0.83	0.45	**0.37**	1.00



(a) Correlation (b) Bias (c) Mean Absolute Error

Figure 2: Performance of QuesIndep models contrasted against Baseline-TC

agreement of two human experts ($r = 0.81$, $bias = 0.14$) and shows that the model may be used in a real world setting. The Q_{all} and Q_{unseen_tr} sets follow the same trends and we discuss them together. In comparison to the testcase baseline, the question independent model does much better on each of the metrics - r, bias and MAE. When compared to the question specific models (**QuesSpec-All**, **QuesSpec**), the question independent model provides comparable performance in their r. However, it shows a higher bias and MAE. This shows that some scaling issues do creep in while building question independent models and this aspect may further be improved. However, the difference in MAE for question independent and question specific models is not very large and varies between 0.12 and 0.25 points. We find that normalization helps both in terms of r (0.04) and bias ($0.04 - 0.07$). These observations are as expected.

In summary, we find that the question independent models rival expert grades, largely outperform test case measures, and show higher bias as compared to question specific models. Furthermore, normalizing the structurally invariant features does help build better models.

We now discuss the errors on individual questions. We compare the three error metrics on our best question independent model **QuesIndep-N1** with those on the baseline (Table 3). We find that the r and MAE of the question independent model outperform that of the TC baseline by 0.15 points on an average. This is also distinctly visible in Fig 2a and Fig 2c (X-axis represents the error metric of Baseline-TC and the Y-axis represents the error metric of the models, **QuesIndep-N0** and **QuesIndep-N1**). We see in Fig 2c that the performance on MAE by **QuesIndep-N0** on 3 questions does worse than the baseline. However, the performance on these 3 questions improves on using the model with normalization, reinforcing the utility of normalization in the learning process.

When we compare biases, we see that the question inde-

pendent model performs better than the test case baseline on 13 of the 19 questions (Fig 2b and Table 3). Out of the 6 questions that underperform: 2 are from the Q_{seen_tr} set and 4 from the Q_{unseen_tr} set (biases marked in boldface). A reason for this under performance could be the normalization technique not being able to get the distances for these questions on the same scale perfectly well. The simple linear scaling demonstrated through our techniques may be replaced by more sophisticated non-linear scaling, warranting a more detailed study.

To conclude, the question independent model consistently outperforms the baseline on correlation, which is expected. As a result of possible normalization issues, the model underperforms in bias in a few cases. However, in no case is the MAE affected and we see it consistently outperform the baseline, demonstrating the utility of the learning technique we introduce in this work.

4. DEPLOYMENT

We deployed the **QuesIndep-N1** model on *Automata*, our online programming evaluation platform. A leading software product company having more than $10,000$ employees wanted to use this product for their 2015-2016 hiring cycle of software engineers with 0-2 years' experience. The company wanted to deliver 13 questions developed by them on this platform. Having released the 13 questions in June 2015, we had 68 responses on an average in the *good set* across the questions within 3 weeks. By the first week of July, we had the question independent models live and evaluating responses to the questions. This is in stark contrast to the $4-5$ month process it would typically take to develop question specific models per question. Till December 2015, a total of 3176 candidates had been evaluated on this platform. The company's default shortlisting criterion was to consider candidates passing at least 75% of their test cases, which was followed by an interview where the final hiring decision was made. In this hiring cycle, they shortlisted 468 candidates who had scored 4 or 5 as predicted by the **QuesIndep-N1** model. Had the hiring been done only based on the test-case cut-off, 391 candidates would have made it through. 384 candidates were common to these two sets. Among the 84 candidates $(468-384)$ interviewed by the virtue of **QuesIndep-N1**, the interviewers found the grades of 76 respondents match their evaluation. Hence, 16.5% candidates $(\frac{76}{76+384})$, a sizeable proportion, would not have made it through to the interviews had the criterion been only the test case score. On the other hand, of the 7 candidates who were graded low but had a high testcase score, all were graded 3 by the model, had an average testcase score of 76.3% and seemed to be at the cusp of being graded 3 or 4 on visual inspection by the interviewers. At the end of the process, 136 of the shortlisted 468 candidates were offered a job. 46 of these 136 (33%) were signaled by **QuesIndep-N1**. This case study demonstrates the successful deployment of **QuesIndep-N1** models in a high-stakes assessment, helping save man-hours which would have otherwise gone into designing question specific models. In a separate study done on candidates who have completed 6 months in job, we also found the predictions of **QuesIndep-N1** to correlate significantly $(r = 0.8)$ with the manager ratings on job performance.

5. CONCLUSION AND FUTURE WORK

We present in this work a system to grade computer programs using question-independent supervised machine learning models. Building on the grammar of features we introduce in our previous work [25], we engineer *semantically invariant* features which maintain their structural relation with the labels across questions. This is possible because we identify a set of 'good' responses to a question automatically. We show that a single model learned on such features, derived from the responses to a few questions, is able to predict the grades of responses to questions not seen in the model's training. Further, we show that the performance of such a model clearly outperforms extant evaluation techniques, rivals the performance of question-specific models and the consensus of human experts. In doing so, we successfully demonstrate how to grade responses to questions without *any* manually assigned labeled samples. Through a case study, we show how practically efficient the system is when deployed to recruit software engineers, saving significant time and resources in the process.

We see a window for further research in the normalization techniques we lay out and the transformations to create a larger set of structurally invariant features. The relationship of accuracy with the number of train problems, size of data set and the number of good codes for new problems can be studied further. Other learning techniques which model rankings better, like ordinal regression, can also be explored. The automatic identification of functionally correct responses, which we demonstrate in this work, can be extended to other areas as well like open response mathematical questions and electronic circuit solving, where a numeric comparator judges the final answer of a series of equations. It also applies to evaluations whose outcomes are human intelligence tasks (such as evaluating whether a webpage is aesthetically designed); the identification of correct responses then can be crowd-sourced in real time to produce a high quality *good set*. These observations promise interesting opportunities for research in such areas, all of which will help the holy grail of designing fully automated teaching assistants. We look forward to the learning from this work resulting in similar accurate, scalable grading systems being designed in other domains.

6. REFERENCES

[1] Automata. Aspiring Minds http://www.aspiringminds.com/technology/automata.

[2] E-rater. ETS http://www.ets.org/research/topics/as_nlp/writing_quality/.

[3] Intelli metric. Vantage Learning http://www.vantagelearning.com/products/intellimetric/.

[4] Speechrater. ETS https://www.ets.org/research/topics/as_nlp/speech/.

[5] Svar. Aspiring Minds http://www.aspiringminds.com/technology/svar.

[6] V. Aggarwal, S. Srikant, and V. Shashidhar. Principles for using machine learning in the assessment of open response items: Programming assessment as a case study. In *NIPS Workshop on Data Driven Education*, 2013.

[7] J. Baxter. A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine Learning*, 28(1):7–39, 1997.

[8] J. Bernstein, A. Van Moere, and J. Cheng. Validating automated speaking tests. *Language Testing*, 2010.

[9] M. Birenbaum and K. K. Tatsuoka. Open-ended versus multiple-choice response formatsâĂŤit does make a difference for diagnostic purposes. *Applied Psychological Measurement*, 11(4):385–395, 1987.

[10] H. M. Breland. The direct assessment of writing skill: A measurement review. *ETS Research Report Series*, 1983(2):i–23, 1983.

[11] J. Burstein, L. Braden-Harder, M. Chodorow, S. Hua, B. Kaplan, K. Kukich, C. Lu, J. Nolan, D. Rock, and S. Wolff. Computer analysis of essay content for automated score prediction: A prototype automated scoring system for gmat analytical writing assessment essays. *ETS Research Report Series*, 1998(1):i–67, 1998.

[12] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[13] H. Daume III and D. Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*, pages 101–126, 2006.

[14] E. L. Glassman, J. Scott, R. Singh, P. J. Guo, and R. C. Miller. Overcode: Visualizing variation in student solutions to programming problems at scale. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 22(2):7, 2015.

[15] J. Huang, C. Piech, A. Nguyen, and L. Guibas. Syntactic and functional variability of a million code submissions in a machine learning mooc. In *AIED 2013 Workshops Proceedings Volume*, page 25. Citeseer, 2013.

[16] A. S. Lan, D. Vats, A. E. Waters, and R. G. Baraniuk. Mathematical language processing: Automatic grading and feedback for open response mathematical questions. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*, pages 167–176. ACM, 2015.

[17] N. Meinshausen and P. Bühlmann. Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4):417–473, 2010.

[18] L. Pappano. The year of the mooc. The New York Times (Accessed: 2016-2-2).

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[20] K. Rivers and K. R. Koedinger. Automatic generation of programming feedback: A data-driven approach. In *The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013)*, page 50, 2013.

[21] V. Shashidhar, N. Pandey, and V. Aggarwal. Automatic spontaneous speech grading: A novel feature derivation technique using the crowd. In *Proceedings of the Conference of the Association for Computational Linguistics*. ACL, 2015.

[22] V. Shashidhar, N. Pandey, and V. Aggarwal. Spoken english grading: Machine learning with crowd intelligence. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2089–2097. ACM, 2015.

[23] R. Singh, S. Gulwani, and A. Solar-Lezama. Automated feedback generation for introductory programming assignments. In *ACM SIGPLAN Notices*, volume 48, pages 15–26. ACM, 2013.

[24] V. Southavilay, K. Yacef, P. Reimann, and R. A. Calvo. Analysis of collaborative writing processes using revision maps and probabilistic topic models. In *Proceedings of the Third International Conference on Learning Analytics and Knowledge*, pages 38–47. ACM, 2013.

[25] S. Srikant and V. Aggarwal. A system to grade computer programming skills using machine learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1887–1896. ACM, 2014.

[26] S. Thrun. Is learning the n-th thing any easier than learning the first? *Advances in neural information processing systems*, pages 640–646, 1996.

[27] C. Vleuten, G. Norman, and E. Graaff. Pitfalls in the pursuit of objectivity: issues of reliability. *Medical education*, 25(2):110–118, 1991.