Clustering LaTeX Solutions to Machine Learning Assignments for Rapid Assessment

Sindy Tan and Finale Doshi-Velez
Harvard University
sindytan@alumni.harvard.edu
finale@seas.harvard.edu

Juan Quiroz
Sunway University
juanq@sunway.edu.my

Elena Glassman
UC Berkeley
glassman@alum.mit.edu

ABSTRACT

Math assignments in machine learning (ML) classes allow students to gain competency in performing derivations. These derivations are graded by hand, which is a time-consuming process. We present our on-going work on clustering derivations so that they can be graded by cluster, rather than one at a time. This paper describes our current algorithms and interface for merging mathematically equivalent expressions, a first step toward the goal of clustering solutions.

1. INTRODUCTION

Demand for machine learning (ML) education is rising rapidly. While writing code to analyze data and train models is an essential pedagogical component of ML courses, equally essential are math assignments that allow students to gain competency in performing derivations. Unlike programming assignments that can be empirically verified using a battery of teacher-chosen test cases and unit tests, these student-written ML derivations contain sequences of non-trivial mathematical expressions, often written in LaTeX. There is little prior work on automatically checking LaTeX expressions and derivations, let alone those of the length and complexity commonly seen in even basic undergraduate ML homework assignments.

As a result, grading can be a time-consuming manual effort. In large classes and Massive Open Online Courses (MOOCs), this grading process can require building a large staff of sufficiently qualified teaching assistants to each perform many hours of work grading solutions one at a time. Not only is time spent grading time spent away from students, but human graders are subject to fatigue and may make mistakes or drift in their interpretation of a rubric over time.

One way to reduce time spent grading is to create tools for powergrading, i.e., clustering solutions so that teaching assistants can grade solutions one cluster at a time rather than one solution at a time [1]. In the powergrading paradigm, the teaching assistant only needs to check if all the solutions in the same cluster should receive the same grade, and then assign the grade. Checking clusters is faster than grading, thus reducing the overall time spent grading [5; 3]. By assigning all solutions in a cluster the same grade, we hope to also achieve greater grading consistency. Importantly, even if not all solutions can be clustered, being able to cluster a large fraction of the solutions can still result in efficiency and consistency gains.

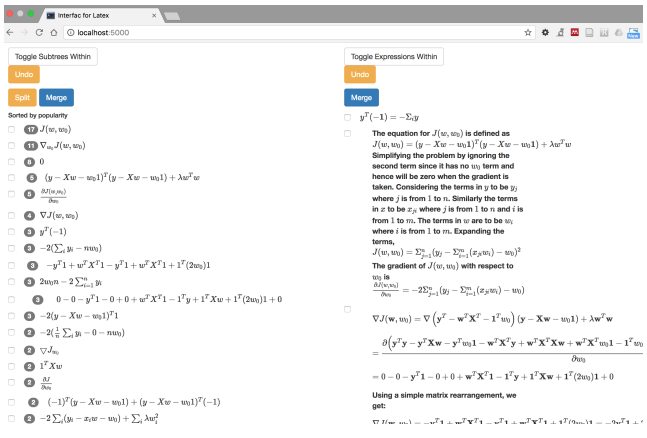

Figure 1: The interface for reviewing LaTeX expressions and solutions for a machince learning (ML) homework problem.

Clustering LaTeX derivations is a large and challenging task. In this paper, we share on-going work focused on an intermediate problem: clustering expressions, or single lines of derivations. There are many different correct ways of writing mathematically equivalent expressions, and students may use multiple ways of denoting mathematical operations, e.g., differentiation. Similarly, each student's choices of symbols, like variable names in programs, suggest a certain semantic meaning, but they may still vary. For example, for problem 2 in our dataset, the grader can safely assume that c_0 and C are the same constant when part of an expression that adds a constant to the term $-\frac{1}{2}w^T w\alpha$. In other words, $-\frac{1}{2}w^T w\alpha + c_0$ and $-\frac{1}{2}w^T w\alpha + C$ can be considered equivalent in answers to problem 2, though that may not be true in general. The clustering is itself driven by machine learning; we design a back-end that re-clusters inputs given corrections from the teaching assistant via the front-end (Fig. 1). From our initial exploration on the intermediate problem, we find that even clustering expressions from raw source can be challenging. We also notice that while expression clustering is a necessary sub-task for clustering solutions, there will still be many additional steps required to cluster solutions. Still, our initial results suggest that this direction may be useful for clustering derivations in the future.

2. RELATED WORK

Our work is similar in spirit to prior work on providing feedback at scale for programming assignments [4; 3; 8]. These systems group together identical code segments so that the teaching staff can understand the different ways in which students completed assignments; correctness is checked automatically by running the code against various unit tests.

Nontrivial mathematical assignments that require nontrivial symbolic answers or derivations present a similar but distinct set of challenges. Lan et al. [5] cluster Matlab answers to open response math questions using a bag-of-expressions model. Responses are collected in Matlab, a symbolic format that is more structured than raw LaTeX.

Given that there are multiple ways to express the same mathematical idea, human judgment can be a powerful tool for making sense of and clustering student-written answers based on their semantic, not superficial, commonalities. Interactive clustering (e.g. [7]) is popular in many domains, ranging from organizing personal travels [9] to document collections [6]. Our current work uses a fairly simple version of interactive clustering, but there are many opportunities for intelligent approaches using active learning.

3. DATA EXTRACTION AND PROCESSING

In order to explore the feasibility of clustering LaTeXsolutions, we acquired a dataset of solutions to Machine Learning homework assignments, parsed these solutions, and extracted a set of hand-chosen features.

3.1 Data set

To develop the grading tool, we used student submissions collected over one semester of Harvard's CS 181 Machine Learning course. We received 341 and 219 LaTeX submissions for the first two homework assignments, respectively. Of these 341 submissions, we train on a random subset of 110 submissions that were not empty and could be parsed by our pipeline, described in further detail in the next section. We only analyze these non-empty, parsable submissions so that, after preprocessing, the extracted math expressions are a comprehensive representation of the original student submissions.

3.2 Pre-processing

Students submitted solutions for each homework assignment as a single LaTeX file. Each homework assignment consisted of three problems, the first two of which required a mathematical derivation. The first problem had three parts, which we refer to as 1a, 1b, and 1c, while the second problem had just one part, which we refer to as 2.

These files were parsed and split by problem. Within each problem, the contents were further split into LaTeX "expressions", which is any text entered in LaTeX's math mode. Only standard commands such as "\begin{itemize} ... \end{itemize}" and "$ ... $" can be parsed for detecting separate solution parts and math mode. Our parser can handle standard LaTeX commands but not imported commands from special packages or user-defined commands or symbols. We ignored any non-math text, e.g., explanations of derivation steps in words, that may also be included in the solutions, inline or otherwise. Any expression that did not include an equals sign (=) was also ignored under the assumption that such expressions are most likely to be part of the explanations of derivation steps. The text that surrounds the extracted mathematical statements is informative and may be included in our future clustering work, but for now, we only consider the mathematical statements themselves.

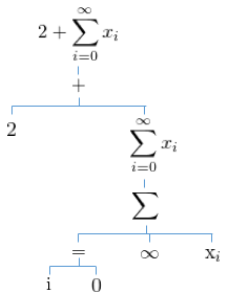

Figure 2: A visual representation of how the expression "$2 + \sum_{i=0}^{\infty} x_i$" is broken down into a math syntax tree (MST).

Each LaTeXexpression was then tranlated, using the *latex2sympy* library, into a canonicalized expression that could be recognized by SymPy, a Python library for symbolic math. As illustrated in Figure 2, this canonicalized expression can be interpreted as a tree, where each node is a mathematical operation or subexpression and each leaf is a number or variable, which we call an "atom." We refer to this as a math syntax tree (MST), because it is similar to the abstract syntax tree (AST) used to describe the structure of source code in various programming languages. Every node of the MST, including the root node and leaf nodes, is considered to be the root of an MST subtree. These MST subtrees represent meaningful groups of atoms and operators that also preserve mathematical relationships, e.g., order of operations.

3.3 Feature Extraction

We manually selected a set of operations and atoms that capture aspects that we expect to be important in correct solutions to the problems in our dataset. These features are listed in Table 1. After extracting all subtrees from the expression MSTs, we check for these features within each subtree.

We use these features to later isolate the relative importance of specific operators and atoms within each problem. For example, if, for one problem, graders are focused on whether each student used the correct signs while, for another problem, graders are focused on summation limits, these features will capture that difference.

Table 1: Hand-picked feature set

Derivative	-1	-2	0	1	2	A
Multiply	w	w_0	w_i	w_j	x	x_{i*j}
Function	y	y_i	y_j	z	bf	$\bigtriangledown$
Integer	J	J_{w_0}	Σ_i	Sum	∇	∇_{w_0}
Symbol	Add	T	X	X_{ij}	Y	a
iszero	j	λ	m	n	∂	$\sum_i$
Power	Tuple	B	C	x_i	vec	i
α	i	$\sum_j$	N			

Our current selection of features is a limitation, since the features are specific to the problems we were clustering.

We leave additional feature extraction that can generalize across a wider range of problems for future work.

3.4 Data handling

We used MongoDB to store all preprocessed solutions, expressions, subtrees, features and their mappings to each other. User interface (UI) relevant attributes such as frequency and circulation status were also stored in the same databases. All derived math objects, such as matrices and vectors, are stored as separate files. The interaction between the UI, database, and backend are as follows: the UI prompts backend computation which updates the database and math objects. Changes in the database update the UI. This architecture simplifies debugging and race condition handling.

4. CURRENT INTERFACE

As part of their grading responsibilities, graders must distinguish between *meaningfully* distinct variations, e.g., incorrect solutions, and *superficially* distinct variations, e.g., correct solutions that follow the same approach but use distinct notation and/or symbols. The interface, shown in Figure 1, allows graders to inform the system of equivalences they recognize at both the expression and solution level.

The current interface renders all the solutions to one problem, and all the unique expressions in those solutions, in a two-column layout. In the left column, the expressions are ordered from most to least frequently used in student solutions. In the right column, entire student solutions, i.e., mathematical expressions and surrounding textual explanations, are listed. When the grader spots two or more items in a column that they consider semantically equivalent in the problem context, they can select these items and click the Merge button. This interface does not yet render the results of the clustering process described in the sections that follow.

5. MACHINE-ASSISTED EXPRESSION CLUSTERING

While graders could manually merge all equivalent expressions and/or solutions to form clusters of equivalent solutions that can receive the same grade, this process could benefit from machine assistance. We now describe a process for assisting this clustering process via interactive clustering: as the teaching assistant assigns expressions to clusters, our system learns parameters to cluster the remaining expressions. The teaching assistant can keep this clustering or continue to make corrections, which will again propagate information to expressions that have not been manually labeled. Knowledge from clustering previous expressions is also retained to reduce the number of labels required for future expressions.

The process of machine-assisted clustering has three main components. First, we must define a distance between two expressions. This distance is parameterized by a weight vector w. Given the weights, we use hierarchical agglomerative clustering to define the clusters. Finally, we define a process for learning the weight vector w that parameterizes the distance function, given human labels indicating which expressions are semantically equivalent.

5.1 Computing Distances between Expressions

To compute distances between LaTeXexpressions, we first define the following terms:

- X: the set of unique expressions in all student solutions to a single problem
- Y: the set of unique subtrees over X
- S: an $|X| \times |Y|$ 1-hot binary matrix from the presence or absence of each subtree from each expression
- Z: the set of features
- S': a $|Y| \times |Z|$ 1-hot binary feature matrix from the presence or absence of each feature in each subtree.
- w: the vector of feature weights

Next we compute an D, an $|X| \times |X|$ matrix of inter-expression Euclidean distances via Algorithm 1.

Algorithm 1 Distance between expressions

function $\textsc{Distance}_e(X, w)$
  $S \leftarrow |X| \times |Y|$ zero-matrix
  $S' \leftarrow |Y| \times |Z|$ zero-matrix
  for $x \in X$ **do**
    for node in $\text{MST}(x)$ **do**
      $S_{x,\text{node}} \leftarrow 1$
  for $y \in Y$ **do**
    for $z \in Z$ **do**
      if z is a feature of y **then**
        $S'_{y,z} \leftarrow 1$
  for $x_1, x_2 \in X$ **do** ▷ only upper triangular needed
    $M \leftarrow (SS')_{x_1} - (SS')_{x_2}$
    $D_{x_1,x_2} \leftarrow (M \odot w)M^T$
  return D

5.2 Clustering

Given the distances between solutions, clustering them is straight-forward. We apply hierarchical agglomerative clustering from sklearn to cluster solutions.

5.3 Learning Distances: Interactive Clustering

As mentioned before, the process of clustering depends very much on our notion of distance. In our case, we have parameterized the distance between two expressions—which governs the rest of inter-solution distance and solution clustering process—by a set of weights w. We start with a uniform set of feature weights w. This w is used to generate an initial clustering. Given a clustering, the teaching assistant can make two kinds of requests: "split" or "merge". A "split" request is when the user moving two or more clustered expressions apart. A "merge" request is the opposite action, moving two or more separated expressions into the same cluster. In our results, we compare between three ways of addressing these requests. We describe each of them below.

Simple update Clusters are presented in order of size. During a merge request, the expression with highest frequency of occurrence persists while all others stop circulating. The frequency of the persisting expression is updated

with the sum of frequencies of all expressions in the request. Ties in frequency are settled by random selection. During a split request, a new cluster is made for each expression in the request.

Simple update with similarity ranking Clusters are presented in order of similarity, as determined by our distance function. The update method itself is the same as the simple update.

Weights-optimized update

When a user makes a merge request between expressions x_i and expressions x_j, the system sets the distance between these two expressions to a target value of 0. The updated target matrix is then sent to a gradient descent algorithm which learns the values of the weights that best match the new target matrix.

We use the $L1$ norm as the loss function for gradient descent as shown in equation 1.

$$loss(w) = \frac{1}{|X|} \sum_{(i,j) \in X} |D_{target(x_i,x_j)} - D_{predicted(x_i,x_j)}| \quad (1)$$

6. RESULTS

6.1 Cluster fingerprints

The first important hypothesis to check is whether comparing expressions based on the subtrees of their math syntax trees allows us to distinguish between equivalent and distinct expressions. Figure 4 shows the subtree fingerprints for each of a small set of expressions. The expressions are manually grouped by our own labeling of similarity.

As seen in all but the fourth cluster of Fig. **??** and Fig. 4, the subtree fingerprints are in most cases sufficient in defining a cluster. The fourth cluster is problematic because there are more variations across expressions than similarities. This is a common issue for clusters with very short expressions, i.e. expressions with few subtrees and thus sparse features. This exploration is informative because it demonstrates that while subtrees may provide an effective means for distinguishing certain human-defined clusters (e.g. clusters 1, 2, 3, and 5), they may not be effective for identifying other clusters (e.g. cluster 4). Knowing that expressions may be challenging to distinguish at the subtree level is important because this challenge will persist for *any* choice of weights w and features (not just those in table 1). One solution to this issue may be to improve canonicalization of atoms in the preprocessing steps.

6.2 Weights optimization

Fig. 5 shows a target distance matrix (343×343) that was generated by calculating the euclidean distances between 343 expressions extracted from all solutions to one problem. We set the first 80 rows of the target distance matrix to a value of 0 to simulate merge requests. Because of symmetry, we also set the first 80 columns to a value of 0. The dark areas represent distances equal to or close to 0. The gray and light areas represent expressions that are farther apart in distance. The learned distance matrix shows that the optimization learning appropriate weights to get close to the target distance matrix (note how the values in the left-most columns of the "Learned" matrix are lighter than the columns of the "Initial").

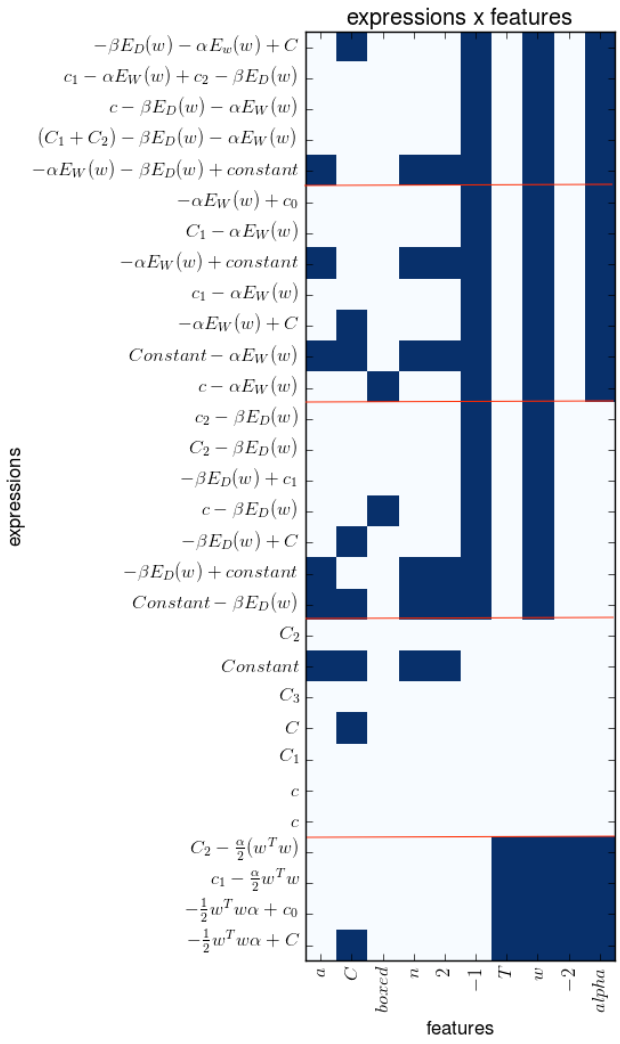

Figure 3: A sample of five expression clusters and their features. The red lines separate expressions into their respective clusters.

7. DISCUSSION AND CONCLUSIONS

Our results above demonstrate that math syntax trees—perhaps with some additional canonicalization—may be sufficient for distinguishing clusters of expressions in many cases. We also showed that our approach of parameterizing distances as weights on features found in subtrees of math syntax trees can be used to make inter-expression distances get closer to matching annotations (and once distance functions are close, clusterings will also be close). That said, while our "Learned" matrix is clearly closer to the "Target" than the "Initial," it is still far from a perfect match. It remains an open direction for future work to determine the best way to featurize expressions such that it is possible to learn distance functions that recover human-desired clusterings.

Specifically, we believe that improving features will likely require moving beyond a simple application of the *latex2sympy*

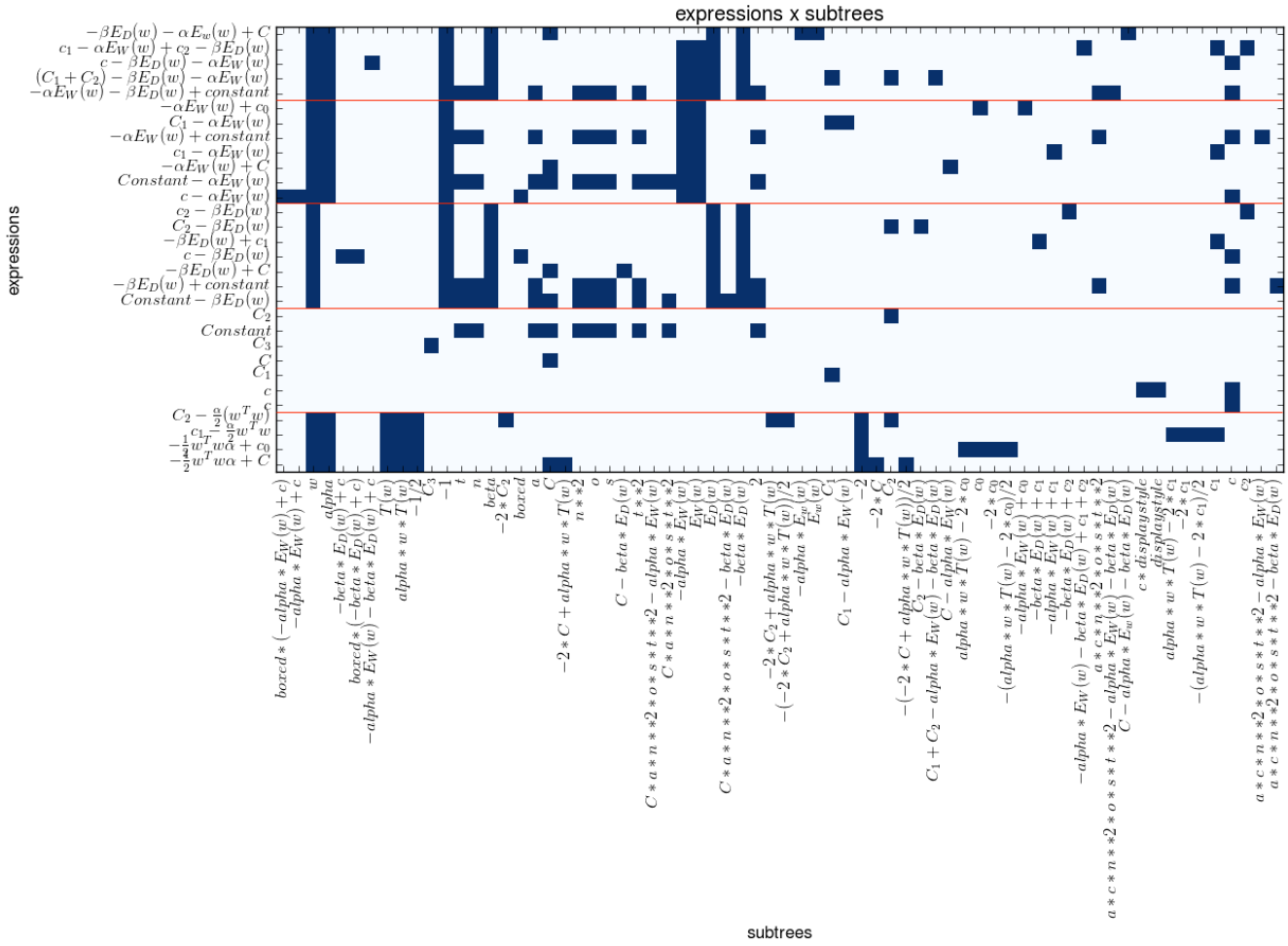

Figure 4: A sample of five expression clusters and their subtrees. The red lines separate expressions into their respective clusters.

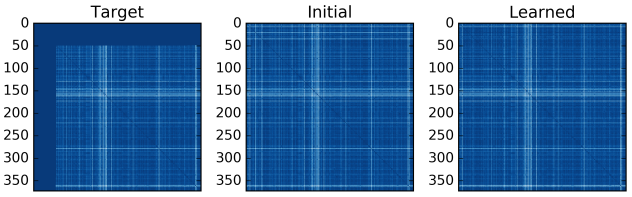

Figure 5: Target distance matrix, initial distance matrix with weights initialized to uniform value of 1, and learned distance matrix after 100 steps of gradient descent.

library. The *latex2sympy* parser is unable to convert a large number of expressions to SymPy, resulting in loss of important information for determining similarity between solutions. One solution is to improve the lexical analyzer and parser of the *latex2sympy* library. In addition, while there is a set of standard commands in LaTeX, the availability of specialized packages and custom commands limits the scope of what can be parsed by our math grading tool. Recent advancements in decompiling visual markup suggest a promising solution [2]. In particular, since there is less variation in rendered math than the markup used to generate it, it would be more manageable to parse decompiled LaTeX than raw LaTeX submissions from students.

Finally, future work on comparing expressions should determine when we get distances "good enough." While it may be challenging to create feature sets such that expressions in the same cluster have perfect similarity, it may be sufficient that they are all sufficiently similar that they get clustered together, and other points are sufficiently dissimilar that they are excluded.

More broadly, there remains the broader question of how to use ways of clustering expressions to help cluster solutions. Again, we will need to define a distance metric, now between solutions. A simple approach would be to treat expression distances as costs, and then compute the cost of the best bipartite match between two solutions as the cluster similarity. However, from an initial exploration of the solution clustering problem, we believe that more sophistication will be needed, and clustering expressions is only a piece of the broader problem. While there are cases—such as those we showed—where students write equivalent expressions in different ways, it is also true that simply the nature of the problem set-up ensure that a lot of expressions are already

equivalent. To effectively cluster solutions, we will have to identify patterns in how expressions are ordered and skipped (rather than a generic bipartite match), as well as integrate information from the surrounding text (which may correspond to a skipped line). Not being able to cluster certain expressions—such as the short ones in figure 4—may not be a practical bottleneck if there are ways to make such lack of expression clusters not contribute overmuch to solution clusters. Once we tackle this objective, we will be closer to our overall goal: creating a tool to reduce the burden of grading and providing feedback for a large number of solutions by clustering the solutions and allowing the teaching assistant to grade only one representative solution from each cluster.

8. ACKNOWLEDGEMENTS

Ryan Bellmore for extracting student submissions for us to use as data.

9. REFERENCES

[1] S. Basu, C. Jacobs, and L. Vanderwende. Powergrading: a clustering approach to amplify human effort for short answer grading. ACL Association for Computational Linguistics, October 2013.

[2] Y. Deng, A. Kanervisto, and A. M. Rush. What you get is what you see: A visual markup decompiler. *Association for the Advancement of Artificial Intelligence*, pre-print on arXiv, 2017.

[3] 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.

[4] S. Gulwani, I. Radiek, and F. Zuleger. Feedback Generation for Performance Problems in Introductory Programming Assignments. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 41–51, New York, NY, USA, 2014. ACM.

[5] 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.

[6] H. Lee, J. Kihm, J. Choo, J. Stasko, and H. Park. ivisclustering: An interactive visual document clustering via topic modeling. In *Computer Graphics Forum*, volume 31, pages 1155–1164. Wiley Online Library, 2012.

[7] M. Rasmussen and G. Karypis. gcluto: An interactive clustering, visualization, and analysis system. *UMN-CS TR-04*, 21(7), 2004.

[8] 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*, KDD '14, pages 1887–1896, New York, NY, USA, 2014. ACM.

[9] C. Zhou, D. Frankowski, P. Ludford, S. Shekhar, and L. Terveen. Discovering personal gazetteers: an interactive clustering approach. In *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 266–273. ACM, 2004.