2017-01-01

A Semi-Automatic Grading Experience for Digital Ink Quizzes

Brooke Ellen Rhees
Brigham Young University

A Semi-Automatic Grading Experience for Digital Ink Quizzes

Brooke Ellen Rhees

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Dan R. Olsen, Chair
Kevin Seppi
Michael Jones

Department of Computer Science

Brigham Young University

ABSTRACT

A Semi-Automatic Grading Experience for Digital Ink Quizzes

Brooke Ellen Rhees
Department of Computer Science, BYU
Master of Science

Teachers who want to assess student learning and provide quality feedback are faced with a challenge when trying to grade assignments quickly. There is currently no system which will provide both a fast-to-grade quiz and a rich testing experience. Previous attempts to speed up grading time include NLP-based text analysis to automate grading and scanning in documents for manual grading with recyclable feedback. However, automated NLP systems all focus solely on text-based problems, and manual grading is still linear in the number of students. Machine learning algorithms exist which can interactively train a computer quickly classify digital ink strokes. We used stroke recognition and interactive machine learning concepts to build a grading interface for digital ink quizzes, to allow non-text open-ended questions that can then be semi-automatically graded. We tested this system on a Computer Science class with 361 students using a set of quiz questions which their teacher provided, evaluated its effectiveness, and determined some of its limitations. Adaptations to the interface and the training process as well as further work to resolve intrinsic stroke perversity are required to make this a truly effective system. However, using the system we were able to reduce grading time by as much as 10x for open-ended responses.

Keywords: education, grading, digital ink, interactive machine learning

Table of Contents

1. Introduction

Teachers who want to assess student learning and provide quality feedback are faced with a challenge when trying to grade assignments quickly. These teachers require grading systems which can minimize grading time and maximize quality of assessment. Existing solutions are limited in at least one of these areas. The grading process is either laborious and slow or it fails to provide clear information on what a student understands. Open-ended assessments are useful when evaluating understanding, but require excessive time to evaluate. Multiple-choice questions minimize grading time, but limit assessment ability.

We created SAIGE, the Semi-Automatic Ink Grading Experience, a new quizzing format which uses digital ink-based questions coupled with interactive machine learning to reduce grading overhead for open-ended quizzes. Figure 1.1 shows an example problem in this format, a simple graphing problem converted to digital ink with the student's response marked in blue. Teachers can manually attach feedback (including a point value) to a student response by using the section on the right, where feedback is created and saved for re-use.

SAIGE uses a hybrid of manual and automated grading using a one-at-a-time grading paradigm. After a teacher has created a question and students have responded, the teacher will open the grading view and receive a single student answer to grade. The teacher can step through student-by student, or by selecting "next ungraded" run an autograder which automatically steps through student answers and assigns them feedback. When the autograder reaches a response it cannot classify, it provides that response to the teacher to grade. SAIGE is designed to overcome the biggest challenges to effective grading, and quickly evaluate a large number of student responses to turn a linear problem into a sublinear solution.

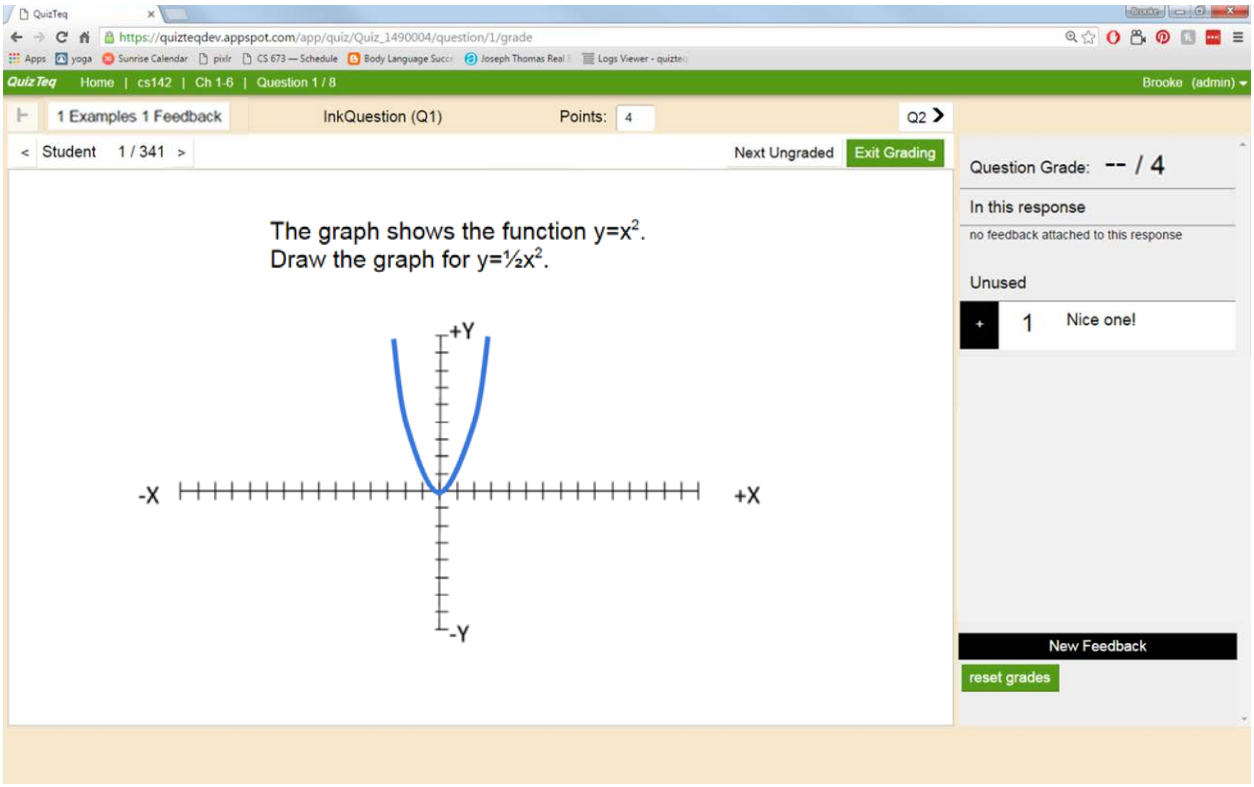


Figure 1.1: A digital ink graphing problem

Current Challenges to Effective Grading

A useful grading system maximizes grading speed, provides consistent grades, and provides a rich testing experience with high-quality questions and effective teacher feedback.

Grading speed can be estimated by total time spent grading all students' answers divided by the number of students who completed the assessment, and is linear with the number of responses. Speed is critical because teachers' and students' focus must move on to new content so grades and feedback quickly become irrelevant [9]. In order to use tests as a teaching tool, test results must be processed rapidly enough to be useful to both teachers and students as they move forward.

Consistent grading involves assigning equal grade values to equivalent work. Grade values include both points and any feedback a teacher may provide for a response. Teachers

benefit when grading variability is removed, because there are fewer inconsistencies that may need to be addressed later and fewer teacher mistakes that would need to be rectified, which would improve overall grading accuracy.

A rich testing experience is one in which teachers can ask high-quality questions to effectively evaluate students' knowledge and provide effective feedback to improve students' understanding. Studies show that higher-education teachers prefer open-ended assessment such as freeform drawing or text response. They value the flexibility of questions and the insights into students' understanding of the material [14]. Effective feedback includes specific comments or guidance from a teacher about a student's work. Teachers who wish to provide insights about students' understanding need a way to communicate those insights. This could include adjusting lesson material or providing specific feedback pertinent to a problem. However, when teachers must provide the same feedback for many student responses, it can quickly become painful.

Thesis statement

A digital-ink assessment program can be created with a semi-automated grading process to improve the teacher assessment experience by including rapid, consistent grading, specific teacher feedback, and high-quality questions.

Project Description

Teachers can use SAIGE to create a set of digital ink questions similar to existing handwritten assignments. Students respond to each question by drawing digital ink strokes on top of the question. Teachers then grade student responses one at a time by attaching feedback (a grade and comments) to each digital ink stroke. After grading a response, the teacher requests the next ungraded student response. The autograder grades the student responses until it finds one that cannot be completely autograded. It gives the incompletely grade response to the teacher for

completion, and the cycle of manual and autograding continues until there are no more answers ungraded.

Semi-automated grading maximizes speed for manual grading while providing consistent, accurate grades. Teachers can give specific feedback that can be applied to matching answers, and digital ink provides a useful format for high-quality questions.

Examples of Use

Figure 1.2 shows two versions of a question about the effect of a catalyst on activation energy of a reaction. On the left is the original multiple choice version, with three options describing a possible effect. On the right is a digital ink version which asks the student to draw a new curve to represent the reaction after adding a catalyst.

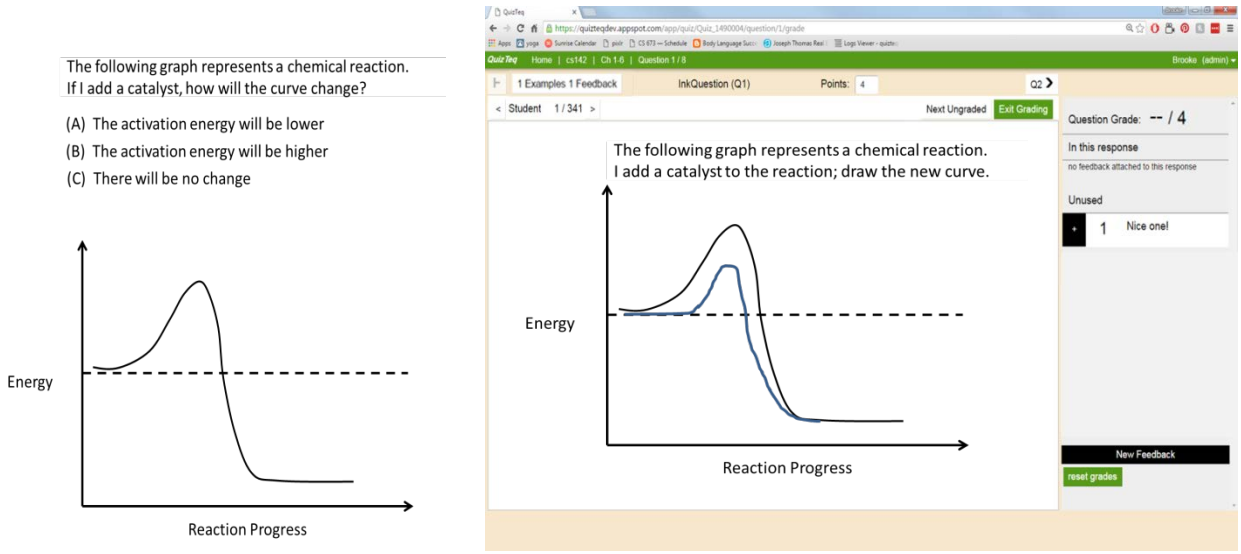


Figure 1.2: Multiple Choice vs Digital Ink for a Chemistry Question

In the digital ink version a student has to know where the curve should fall, with no options to provide hints about the effect on activation energy. The graph for a correct answer has dimensions and shape distinct from other possible responses. A correct answer will have a similar shape to the original graph, but with a smaller peak. An incorrect answer could have a

taller peak or a higher or lower end than the original. These distinct shapes make a graph learnable for algorithmic recognition.

Figure 1.3 illustrates an additional use case. In this second example, the instructions are to underline or circle errors in text. Lines and circles have distinct shapes which are easily recognizable, and in both cases the size and position of strokes are included in the digital ink stroke. SAIGE has no recognition algorithms for understanding programs or text. But SAIGE learns that a mark on the "P" should receive the "Capitalized P" feedback and the mark at the end of the print statement should receive the "Missing semicolon" feedback.

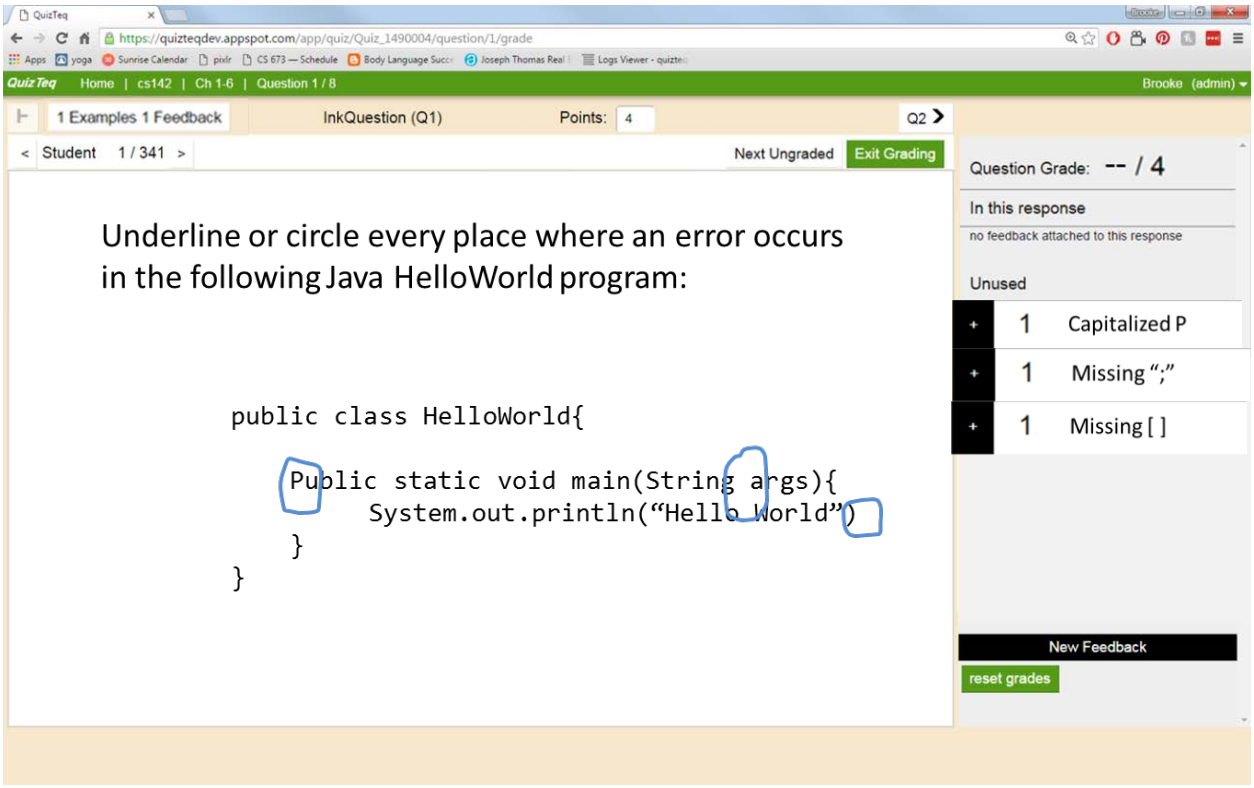


Figure 1.3: Analyzing code for an introductory computer science class.

Using SAIGE, questions can retain their richness and become faster to grade using recyclable feedback and an autograder to work with the teacher. These key advantages of SAIGE stem from its use of digital ink technology combined with interactive machine learning

techniques. We use the information a digital ink stroke contains about location, shape, and size to generate features for training a machine learning algorithm.

Interactive Machine Learning

A machine-learning approach allows teachers to quickly train the algorithm to recognize answers and apply appropriate feedback and grades using digital ink strokes as training data. In the programming example above, a teacher selects a stroke and then attaches feedback to it. The stroke provides a set of features to describe shape and location, and the feedback identifier is a class label. We use these to train a classifier to recognize whether a stroke's shape and location matches any class of strokes. Thus the SAIGE autograder is a recognizer which maps strokes to feedback items to grade a response. Grading then is a matter of algorithmic classification, and is iteratively improved with incremental teacher input using student responses.

In addition to just using student data, teachers have the option of creating examples to provide a base set of expected strokes for the answer to a question. These labeled strokes are used as additional training data for computer evaluation of answers, so the computer grading process can begin immediately upon student submission. Stroke comparisons allow stroke classification by determining from training data the closest match within a threshold. Recognizable answers are assigned a grade. Unrecognized answers are sent to the teacher for manual classification. Thus grading is reduced to a cycle of training and matching between human and computer.

Since accuracy is so important to grading, we tested two different machine learning classifiers, k-nearest neighbor and version space, to determine which performs best on live data in an IML environment for digital ink strokes.

Metrics for Validation

In order to measure SAIGE's success at overcoming the grading challenges, we use four metrics. We measure the required number of training examples to completely grade a question. We use both absolute number of examples and whether the training is asymptotic over time (number of examples vs autograded responses over time). We also count the total number of classification errors per question to measure consistency and uniformity. We measure time to grade within SAIGE to compare speed between grading manually and semi-automatically.

Number of Training Examples

We want to limit required training to <10% of the total responses. If SAIGE is too much work to use effectively, a teacher would never adopt the system. We consider 10% of expected manual grading an acceptable amount of work for classes with more than 100 students. Using ten or fewer examples is an acceptable limit for classes smaller than 100 students. Figure 1.4 illustrates the amount of work required to grade a question by hand (red) and semi-automatically (green). As the figure shows, grading a fraction of the total responses greatly decreases a teacher's workload. Even for a class of 400 students, this limits grading to only 40 responses, which is a significant difference in time and effort for a teacher who would otherwise be grading manually.

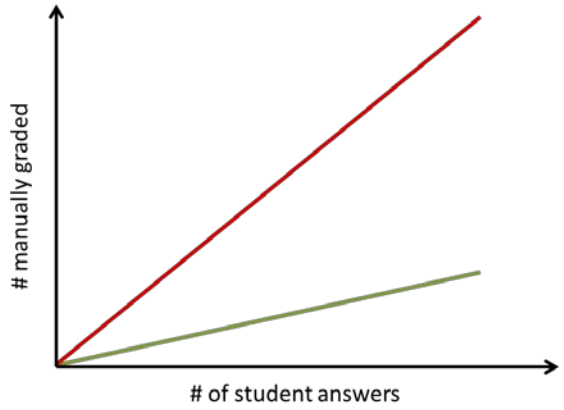


Figure 1.4: The basic amount of work required to grade a question manually or semi-automatically.

Asymptotic Training

At some point, we want to never have to grade again, otherwise grading will remain a linear problem even if we reduce the workload to 10%. Unless grading can become fully automated, a teacher would never reuse a question for a future class because the grading continually needs to be updated. Asymptotic training means sublinearity with regards to grading time and effort. We calculate this trend by looking at the number of required examples (manually graded) against the total number of student responses, as shown in Figure 1.5. The graph illustrates the improvement of asymptotic grading time versus linear grading time. The red line represents the amount of work to manually grade all the responses, the blue line is a reduced amount of work from semi-automatically grading, and the green represents the ideal experience.

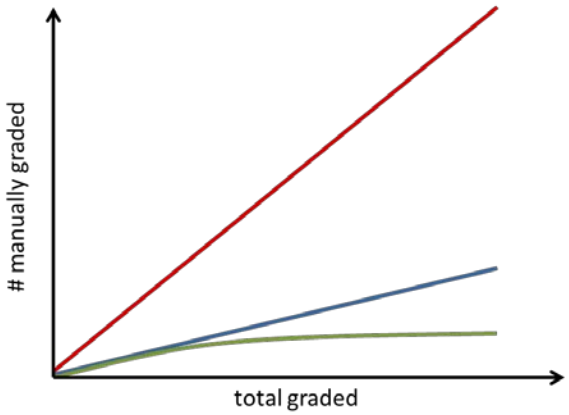


Figure 1.5: The amount of work required to grade a question with asymptotic training.

We measure this by comparing graph of manually graded examples to the linear cap. This gives us a sense of how closely we approach asymptotic training. This metric is related to but distinct from counting the total number of training examples because it caps the number of total examples for large classes, making this a scalable solution. If SAIGE has successfully learned the problem, after a certain number of examples it does not require additional training. For

teachers, this would also mean it becomes possible to recycle questions in the future without having to retrain by grading again.

Classification Errors

In the grading world, the ideal is a perfect grader, because then there would be no need to make corrections to grading mistakes and no additional confusion for students. In reality human graders rarely, if ever, are 100% accurate. Thus although classification errors which would cause incorrect grading are largely unacceptable, we accept a small margin of error. We require that the maximum number of errors be no more than 2% of the total number of autograded responses. This allowance accounts for some errors due to human mistakes and unavoidable classification confusion.

Actual Grading Time

This metric is intended to gauge how much real time is saved by semi-automatically grading large numbers of answers. If using SAIGE requires more than a reasonable amount of time to grade a large class, then adoption by teachers would naturally be low. A teacher should have to spend an average of 30 seconds or less per response, including time waiting for SAIGE to autograde. This allows some time to consider a response and attach necessary feedback, considering that some responses may take more time than others to grade.

2. How SAIGE works

SAIGE combines digital ink and interactive machine learning to create a new grading experience. This chapter outlines the three main elements of SAIGE: the user interface, the machine learning, and the network and data structures.

User Interface

The main SAIGE interface is a question canvas where teachers can edit a question, students can draw their responses, teachers can grade the responses, and students can view the feedback. What follows is an example which discusses the interface in depth over the life cycle of a question, which includes creating the question, taking the quiz, grading the question, and viewing the results.

Creating a Question

In this example, the teacher decides to create a simple matching exercise for the students. Creating a question uses the editor shown in Figure 2.1. Using the panel of buttons on the left and the media library on the bottom of the screen, the teacher edits the question much like editing a PowerPoint slide. After the question has been created, the teacher can also choose to create a new question, add an example, and adjust the number of points available for the question. When the quiz is complete, the teacher opens the quiz for students to take.

Figure 2.1: Teacher Creating an Ink Question

Taking the Quiz

The quiz view for the student is a simplified version of the question page. They have the option to draw, erase, or move a stroke, navigate between questions, and exit the quiz. Figure 2.2 shows the question after the student has finished drawing the answer. After all the students complete the quiz, the teacher grades the student responses.

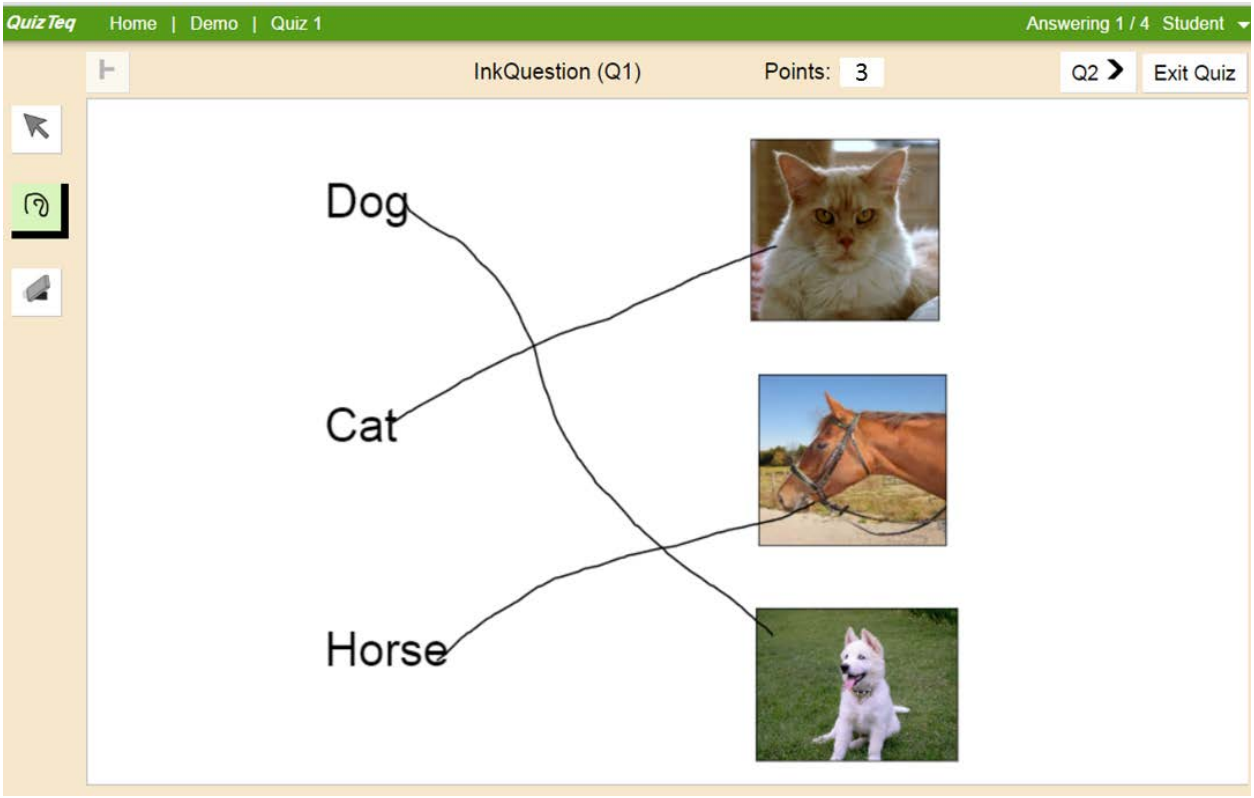


Figure 2.2: Student Taking an Ink Question

Grading the Question

Figure 2.3 shows the layout of the grading interface. When the students have finished taking the quiz, the teacher closes it to prevent further changes from the students. The teacher then opens a question and is presented with a student's response on the canvas. The response includes grade indicators on each stroke to show whether the stroke is ungraded (yellow question mark), received a positive grade (green plus), or received a negative or zero grade (red minus). The teacher then grades the response by attaching feedback to ungraded strokes. In the example in Figure 2.3, the student response has been partially graded by the autograder, with two strokes recognized and one unrecognized. On the right is a feedback panel where feedback is created and edited. The teacher attaches the feedback for the ungraded stroke by clicking on the stroke and then either selecting an unused feedback item from the panel or creating a new feedback item. In

this case, the feedback for the ungraded stroke ("Correct – dog") was created previously, so the teacher recycles it from the panel.

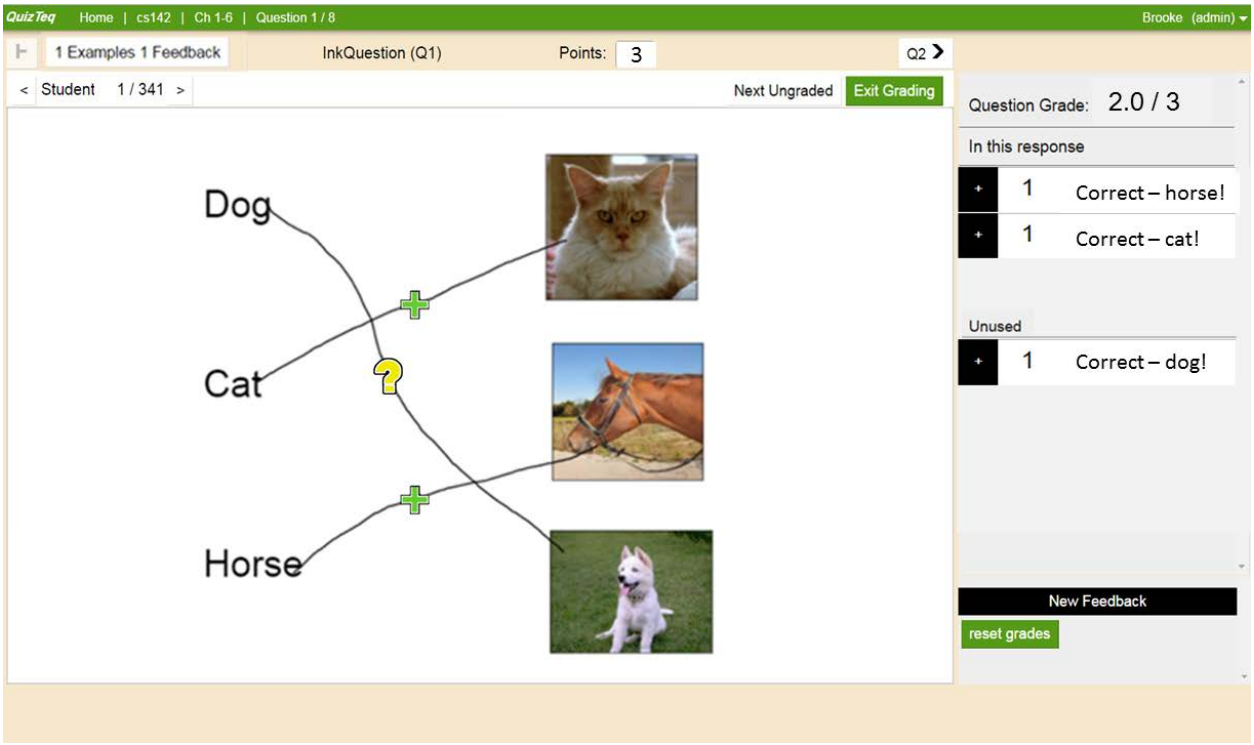


Figure 2.3: Teacher Grading an Ink Question

When finished grading this response, the teacher presses the "next ungraded" button. At this point SAIGE autogrades as many of the remaining ungraded or partially graded responses as it can recognize. If there are strokes within a response the SAIGE does not recognize, it stops autograding and gives the response to the teacher for completion. When all the students have received a grade, the teacher can exit the question or move to a different question. When finished grading, the quiz is available for students to view.

Viewing the Result

Students can view the question and see what feedback they received in the interface in Figure 2.4. The student can click on a stroke on the canvas to see which feedback the stroke received.

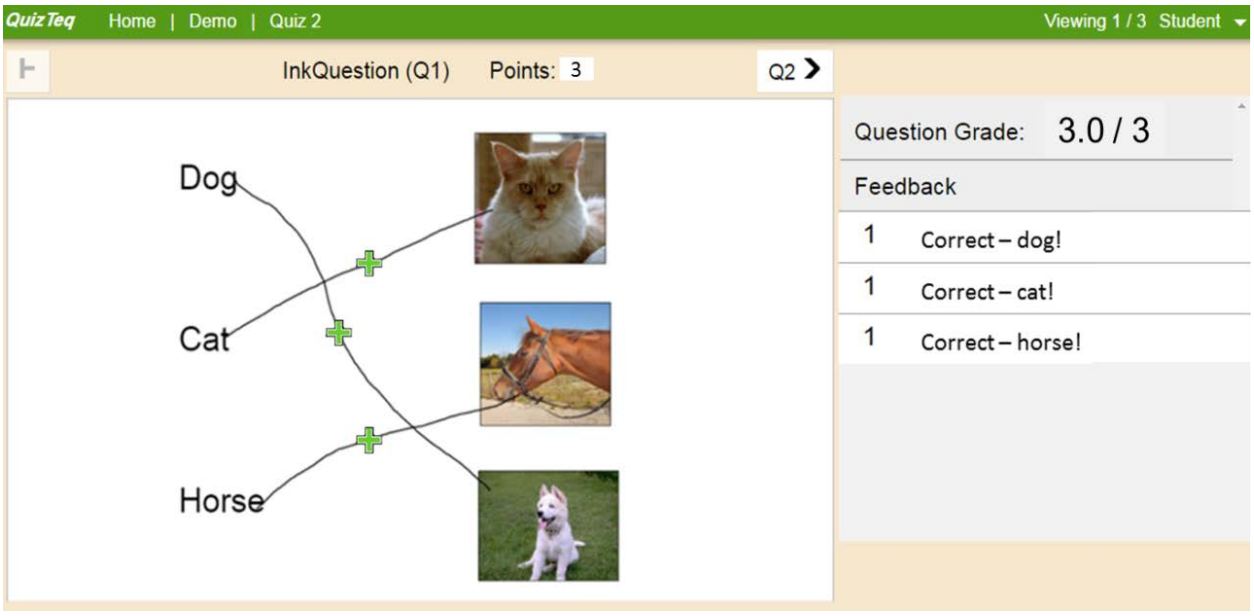


Figure 2.4: Student View of Graded Question

The benefit of this interface is its uniformity and its single-response grading paradigm. The uniformity of the interface made it faster to build, allowing technical exploration to happen sooner. The one-at-a-time grading paradigm is natural for teachers to grade and requires very little explanation. The 1:1 stroke-feedback paradigm allowed very granular feedback, and also created a simple basis for the machine learning by limiting classification to classifying a single stroke at a time.

Interactive Machine Learning

Figure 2.5 illustrates the incremental training system. Each answer a teacher grades adds labeled training data to SAIGE. After the labeled strokes are processed and added as training examples, SAIGE autogrades the remaining student responses until it reaches one it cannot fully recognize. It then hands the response to the teacher to request further training, and the cycle continues until no ungraded strokes remain.

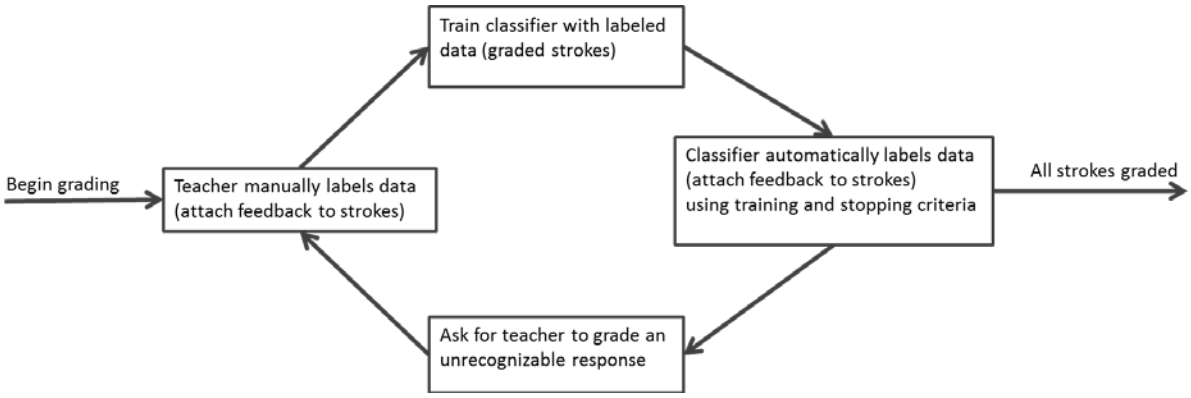


Figure 2.5: Interactive Machine Learning for Grading

Using an incremental training approach poses different challenges than traditional machine learning since the training data within the classifier is constantly changing. Like in any machine learning problem, it is necessary to consider classifiers carefully since not all would be well-suited to the task. In this constantly-changing environment, some classifiers would have to be reconstructed each time training data is added, which is not an ideal interactive experience when that takes additional time.

In addition, unlike traditional machine learning, SAIGE begins classifying as soon as it receives a single training example. In order for this to happen, there must be some kind of generalization rule in place which allows SAIGE to assume an acceptable distance between an unlabeled stroke and training examples in order to assign it a class. This generalization rule can be considered a measure of uncertainty, such as those used in active learning approaches with uncertainty sampling [24]. Traditional machine learning algorithms look at real examples and extract reasonable classification rules based on the training data. SAIGE has such limited training data that some amount of extrapolation is required to achieve the same result. Figure 2. shows the challenges that generalization poses for incremental machine learning. In this example, the dots represent a single example, with black and white to indicate classification. The

generalization rule for this example is that if neighboring features fall within a given range of a training example (indicated with a box), they receive the same classification as the training example. Example (A) is a case of undergeneralization, where SAIGE will require an excessive amount of training to learn the problem. Example (B) is a case of overgeneralization, where SAIGE will extrapolate too much from the training data, resulting in misclassification errors. The ideal is a rule which will generalize on training data just the right amount to balance amount of training with number of errors.

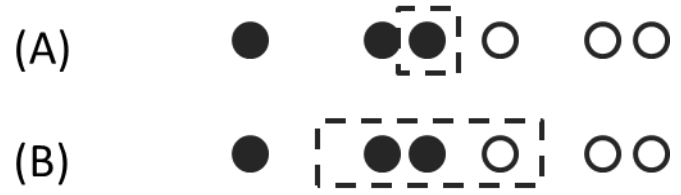


Figure 2.6: Overgeneralizing vs undergeneralizing on training data

Given some generalization rule that defines an allowable generalization distance, SAIGE can do some classification work without additional training data. However, until SAIGE is fully trained, at some point it reaches the limit of what it knows from training data and the generalization rule. At this point, it has to stop and ask for more training data. Thus a generalization distance function is “stopping criteria” for when SAIGE should stop classifying and receive more data.

We have specific requirements for SAIGE to be a good solution for teachers. It must achieve asymptotic training using up to 10% of the student responses as training data. It must also meet our accuracy requirements of 2% or fewer errors across all student responses. Good generalization is critical for achieving these results. If a teacher is grading a question but SAIGE overgeneralized from the training data, the autograder will complete grading quickly but with false results. When this occurs the teacher is unaware of the errors, and the students would

receive incorrect grades and feedback. If the students recognize the errors and make them known to the teacher, the teacher now has to correct the response and clear the grading for all the students' responses so SAIGE can run the autograder with the new training data. Conversely, if SAIGE undergeneralizes on training data the effect of autograding is diminished by the need for constant grading by the teacher. This means it is crucial to use a good stopping criteria function which generalizes enough to minimize a teacher's workload while asking for enough help to minimize errors.

Data and Network structure

We chose to implement SAIGE as a web application as a way to deal with distributability and data access between users. SAIGE's backend is implemented in Python, and its frontend interface is built in JavaScript. SAIGE communicates between the client and server by making requests using a network protocol, and uses a non-SQL database to store its data in JSON format. The network structure allowed us to easily copy student responses from one location to another for testing to keep the original data clean. Figure 2.7 shows a simple network diagram explaining where specific tasks are performed within the client-server system.

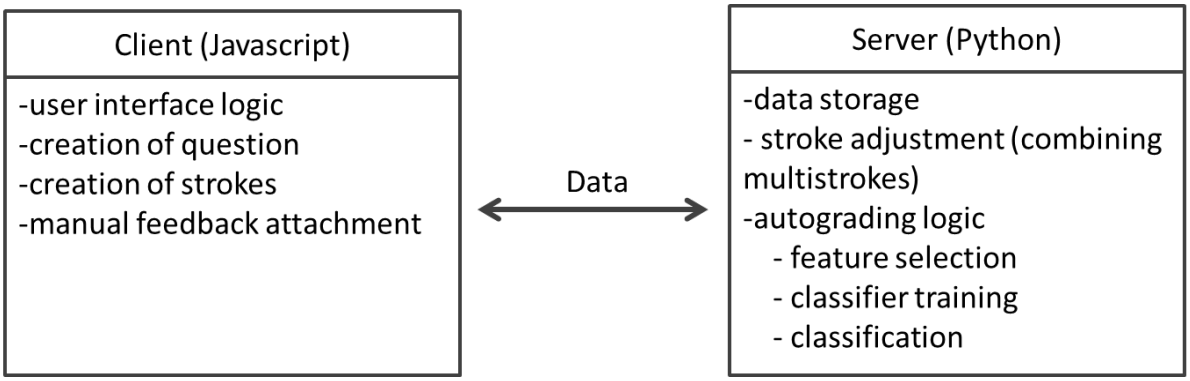


Figure 2.7: Distribution of logic within SAIGE

Data in the form of JSON objects are passed between the server and client. These data provide information about relationships between students, courses, quizzes, and questions, and include objects which store information about questions (background objects and their positions, number of points possible, ordering). Other objects include student answers, which contain a reference to a question, stroke information for a single student, and feedback items. Feedback items are attached to individual strokes within student answers, and contain text comments and a point value.

3. Overview of Technical Challenges

There were four technical challenges we worked with while implementing and refining SAIGE: selecting the best feature set, perversity of strokes, selecting the best classifier, and choosing a reasonable user interface paradigm. This chapter gives a light overview of the technical challenges, as an introduction to more in-depth discussion in later chapters.

Selecting the right features

Feature selection is another critical decision. Using too many features causes the number of training examples to increase due to the curse of dimensionality. Too few features will cause confusion due to missing information, which drives up errors. It was necessary to choose features meaningful to the classification task such as location of the stroke, and to give them proper weight to optimize learning.

Based on preliminary data we discovered most responses will resemble one of three basic kinds of strokes, shown in simplified examples in Figure 3.1. Each has a distinct shape, and each could have features which are more meaningful than others. For instance, a line's primary features are its endpoints, a circle's could be its bounding box and/or center, and a path's could be a sample of the points along the stroke. Other interesting features could include hit locations, in which a stroke encounters a background object. For instance, in the matching problem in Figure 3.1, the stroke hits the two rectangles at several points along the stroke.

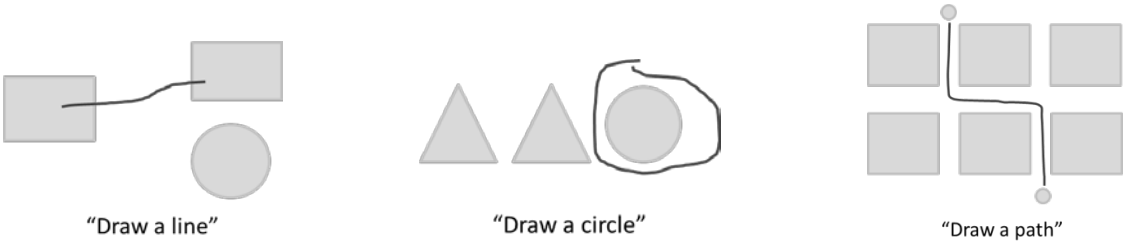


Figure 3.1: Basic Stroke Types

We needed to see if there was an optimal approach to feature sets which would keep examples and errors low. We explored a variety of feature combinations. These included points along the stroke, endpoints, bounding box, and hit locations. Figure 3.2 shows a summary of the features we considered.

Stroke Features	Good For
- N points on stroke ($1) - endpoints - stroke bounding box - hit locations	- nonstandard shapes - straight lines - circles - background context

Figure 3.2: Stroke Features

Perversity of strokes

We discovered that real data from an uncontrolled environment does largely match our hypothesis that student answers are similar. However, the data will also be scattered with perverse strokes. Perversity takes several forms. Some perversity is due to human error: either students fail to follow instructions or they misunderstand the interface. Other perversity is not caused by error, but poses challenges for grading automatically: strokes with the same meaning can take multiple forms, and overlapping strokes can create grading ambiguity for the teacher or the algorithm.

To illustrate this, Figure 3.3 (below) shows one question we asked the students, with a standard response. The instructions are very specific, indicating the students should be underlining (not circling or other marks) and there are four locations to mark.

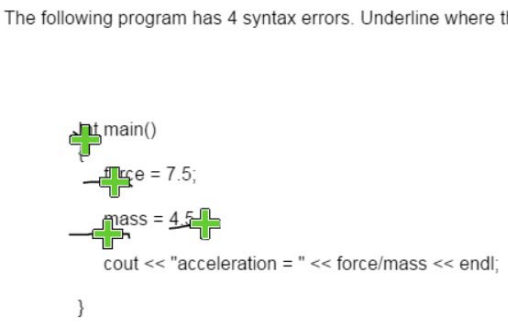


Figure 3.3: Question and response

Figure 3.4 displays one response which contains a large amount of perversity from human error. As the figure shows, occasionally a student would attempt to write in an answer, misunderstanding the interface or not paying attention to the instructions. To SAIGE these marks have no context to give them meaning, and it expects them to receive a grade like any other mark.

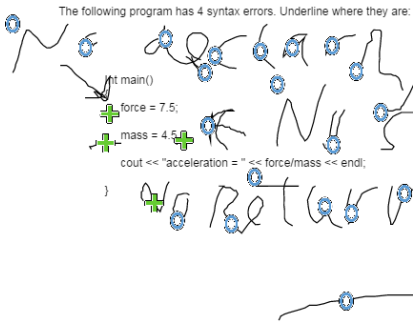


Figure 3.4: Question with perverse strokes

Figure 3.5 shows another response to the same question with a different form of perversity. In this case, an “underline” is not a single straight line, but a thin zig-zag. Such “scribbles” prove challenging when comparing stroke similarity.

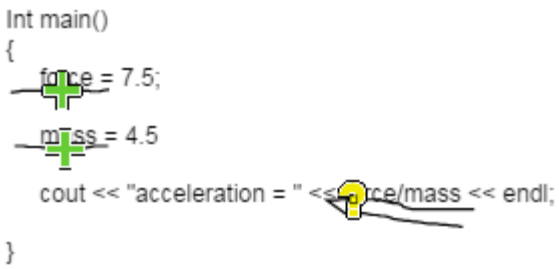


Figure 3.5: A scribble underline stroke

Similarly, we found additional perversity arose in the form of split strokes, where what should be treated as one stroke consists of two or more separate lines. This concept takes two forms, multistrokes and crossouts, as shown in Figure 3.6. Multistrokes are two or more lines with connecting endpoints which could form a single cohesive line (A). Crossouts are two or more lines that cross each other which are intended to represent a single response (B).

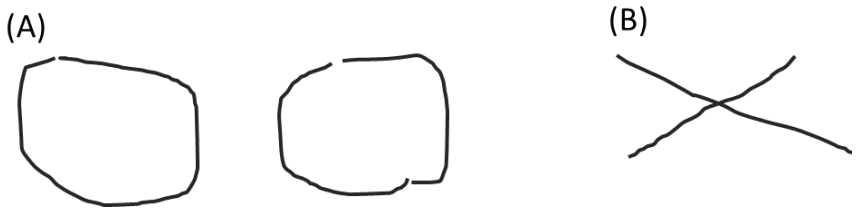


Figure 3.6: Perversity from split strokes in the form of multistrokes and crossouts

We also discovered stroke direction (drawing from start to end) contributed to perversity. Two visually identical strokes which were drawn in opposite directions, shown in Figure 3.7, are very different in feature space, but not to the user.

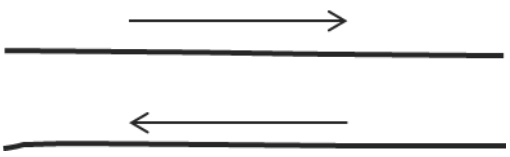


Figure 3.7: Perversity from Stroke Orientation

A different kind of perversity is illustrated in the following example from another question. Students are requested to cross out the error in a piece of code. As shown in Figure 3.8 below, the teacher considered marking the whole line (A) or marking the specific location (C) as "correct" responses. Response (B) should not receive the same feedback as (A) or (C), but it overlaps with (A). The overlap, while visually distinct, creates ambiguity that is challenging to distinguish algorithmically.

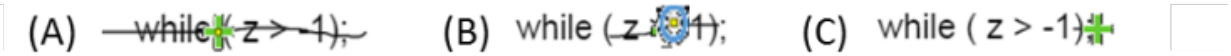


Figure 3.8: Overlapping Responses

Perversity is a problem because more varied data is harder to quickly grade due to the necessity of additional training examples. More training examples also means more chance for human error that could introduce errors in the training data and negatively impact grading accuracy. We explored several options to clean up and work with this perversity automatically to avoid additional manual overhead, with some success.

Choosing a Classifier

SAIGE's incremental training paradigm means that with each manually graded answer, the classifier receives new training that will affect future classification. In the example in Figure 3.9, when the teacher requests the next ungraded answer, each stroke from the current answer is processed into a feature set and those features are added as three labeled training examples to the classifier. Next SAIGE will use what training data it has received to classify as many strokes from ungraded student responses as it can recognize. When it finds something it cannot classify, SAIGE asks for help in the form of further training.

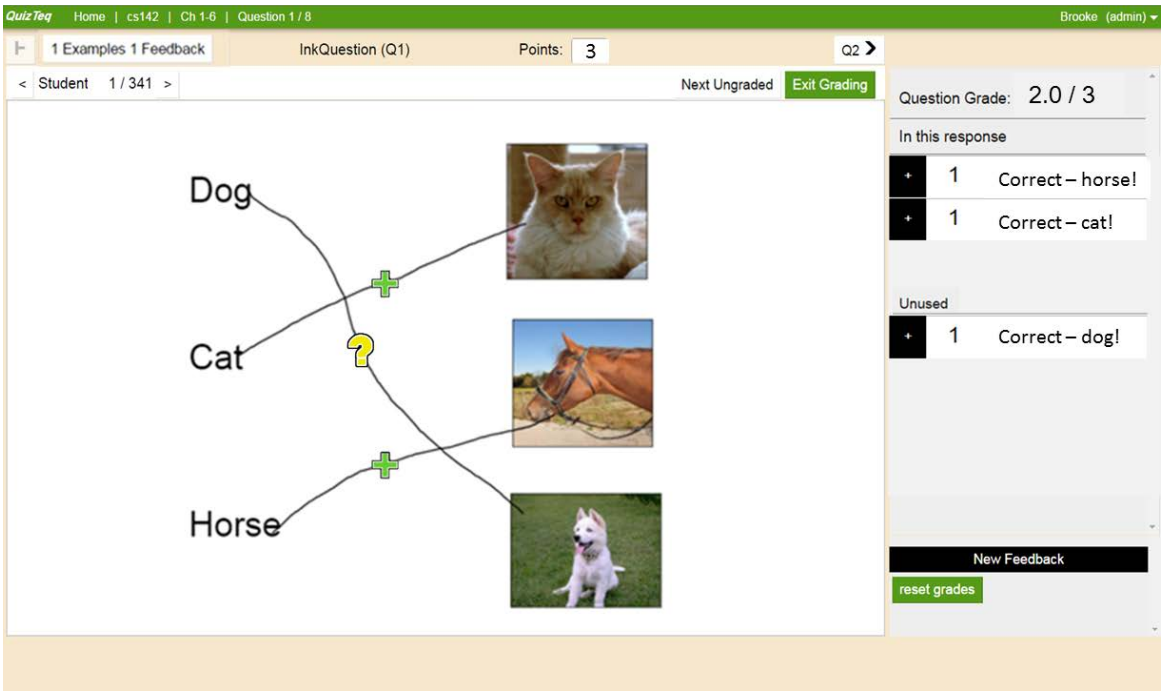


Figure 3.9: Incremental Training

Incremental training such as this poses a challenge not found in standard machine learning. For SAIGE to be effective its classifier had to be able to support incremental training as well as meeting the demands of our metrics. Standard machine learning algorithms have been completely trained on all their training data at the beginning of the classification process, and their goal is to make the best possible decision based on that data. SAIGE also makes the best classification decision possible based on its training data. However, SAIGE begins with no initial training data, and our metrics demand that SAIGE stop asking for training as soon as possible to maximize automation. This means that SAIGE attempts to classify accurately but with 10-15 training examples rather than the 1000-1500 examples a standard algorithm would expect. This constraint affected our decision about which classifier to use. To achieve high accuracy and minimal training, we required a classifier that could both generalize well on limited training data and recognize when to request additional training using stopping criteria.

We informally evaluated the viability of several classifiers by discussing the strengths and limitations of each within the constraints of our grading paradigm. The list of classifiers we considered included naïve Bayes, decision trees, k-nearest neighbor, version space, support vector machines, and deep neural networks. We chose not to pursue any classifiers which posed significant disadvantages. After this preliminary selection we chose to implement KNN and version space in order to compare their performance. A more in-depth discussion of classifier selection, implementation parameters, and evaluation methods is found in chapter seven.

UI paradigm

SAIGE was implemented using an incremental training approach for its classifier. Viewing a single response at a time would provide a simple and easy to use experience for teachers based on the premise that standard handwritten responses are viewed one at a time. We decided to implement a "get-one-and-grade-it" interface. In this interface only one response is displayed at a time and grading involved attaching a single feedback item to each stroke within an answer (a 1:1 relationship). This was attractive because it is simple to understand and easy to build.

We explored the limitations of this approach and discovered that this grading paradigm introduces classification challenges.

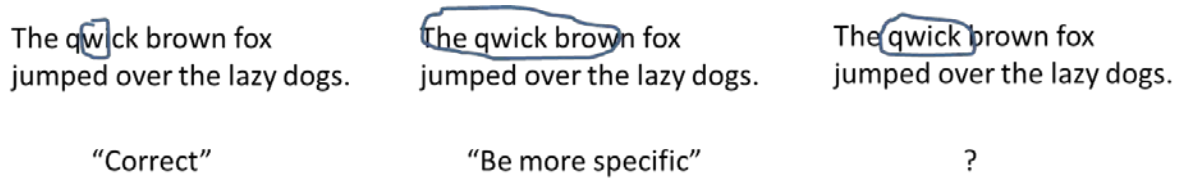


Figure 3.10: Grading Ambiguity

In Figure 3.10 above, a teacher has created a question asking students to mark where the error occurs in the sentence. While grading, the teacher sees the illustrated responses: one very precise, one very imprecise, and one in between. The teacher grades the first with feedback

stating this was the correct answer. The second one is given feedback stating the answer was too imprecise. The teacher sees the third response and decides that this response is specific enough to be marked correct. After grading a few more responses the teacher sees another instance of the third response. This time the teacher marks it wrong, feeling the answer was too imprecise. This example illustrates two issues with one-at-a-time training. Since the teacher sees only one example at a time, the teacher may not remember how a previous response was graded if there is ambiguity to which feedback should be attached. In addition, as the teacher sees more answers, there can be a drift in how they perceive a response to the question, or mistakes while grading due to inattention. Either situation leads to grading inconsistency, which confuses the classifier.

4. Related Work

Solving the grading problem is not a new idea. There are two existing solutions for the grading problem: manual grading with open-ended questions, and automated grading with predetermined responses. An ideal grading system uses open-ended assignments to allow insights and promote communication. This system minimizes time spent grading and provides uniform grades with useful feedback for students. Manual grading and automated grading each have some advantages, but neither achieves this goal. Manual grading is the standard approach to grading open-ended quizzes. Teachers receive more information about students and can provide more feedback for students by using open-ended quizzes. However, manual grading can be laborious and is linear with class size. Manual grading is also likely to cause inconsistency between grades over time. Automated grading uses a graded set of standard responses to grade a student response. Automation provides a vast improvement in grading speed and consistency. However, automated systems are not strictly open-ended and often result in lower-quality questions. There is also much less effective teacher-student communication, since teachers anticipate what students understand rather than addressing specific individual confusions, and opportunities for effective feedback are limited.

Previous computer-related work in this area has focused on two main solution areas: automated grading and providing consistent high-quality feedback. In order to approach this problem from a new technological standpoint, additional areas to explore also include gesture/stroke recognition and interactive machine learning.

Providing High-quality Feedback

One novel approach to grading is the GradeScope program from UC Berkeley [21]. Teachers scan written assignments and use GradeScope to remember feedback as they grade. GradeScope

attempts to solve the grading problem in two ways. It allows open-ended responses, including text, diagrams, and drawing. It also preserves high-quality feedback for re-use so grading time is somewhat reduced. However, teachers must review each response individually, so grading time remains linear in the number of students.

Another grading solution comes from a combined team from University of Washington and Microsoft Research. This solution used text recognition to assist teachers grading short-answer style problems. They analyzed student text and identified similarities to create answer "clusters" for teachers to evaluate [3]. This was a success in several ways; answers were free text, which allowed student insight, teachers did not have to iterate through every answer individually, and feedback was applied to each cluster uniformly. However, there are important limitations to this clustering approach. Clusters were used only for text-based problems. Grading was also entirely teacher-driven after clusters were determined. Teachers had to manually analyze each cluster to determine subdivisions, which takes time.

Automated Grading

One common approach to computer-assisted assessment is multiple-choice questions. The right-or-wrong style of multiple choice questions, particularly the bubble-sheet format, allows rapid uniform grading. Some multiple choice formats allow teachers to attach feedback to the answer options, but feedback is not based on individual responses. The multiple choice format does not allow free-form responses. It also constrains student input to a fixed set of responses, which creates concern among educators about its effectiveness compared with more in-depth question types [5]. Multiple choice bubble sheets bear little resemblance to the subject matter they test, and are often presented in confusing ways to fit the subject matter to the format.

One method of assessment involves using natural language processing to evaluate written answers and provide feedback. The OpenMark assessment system allows authoring of text questions with a set of expected results [11]. Students respond with simple sentences and the program evaluates their response, providing specific feedback on how to improve. OpenMark also provides additional question formats including drag-and-drop with hotspots and simple text matching. This system allows uniform evaluation across student answers with minimal evaluation time. However, it requires initial training which includes indicating all possible answers. The time spent training may outweigh the benefits of minimizing evaluation time. In addition, OpenMark does not allow new teacher insight from student responses. All the feedback is based on predetermined responses, so there is no possibility of responding specifically to novel insight. Although the format for questions and feedback allow more thorough testing, the set of responses remains constrained and the teacher gains no new information from the students.

Stroke Recognition

SAIGE uses digital ink strokes to represent student responses. Our approach to grading requires successful, accurate stroke processing using machine learning. Existing approaches to stroke recognition use a variety of features for comparison.

One stroke recognition approach is the Rubine feature set [17], which has been effectively used for years to compare unistrokes. Another possible feature set is the features from the "$1 recognizer," created by the University of Washington [23]. With over 97% accuracy it can compare unistroke gestures independent of their position, using evenly-spaced points along the stroke as features. This feature set allows a much more flexible set of shapes than the Rubine feature set.

Researchers at the University of Washington built a multistroke recognizer using the same features as the $1 recognizer [1]. It successfully recognizes multistrokes, but has some limitations. Speed becomes an issue as the number of strokes increases, and it struggles with differentiating similarly shaped symbols such as “x” and “y.” A set of researchers at Texas A&M developed an educational writing interface for students to learn a phonetic script. They used segmented raw strokes to reconstruct the individual symbols in the vocabulary [18]. This research allowed multistroke recognition based on an assumed stroke drawing order. The research was focused on a specific recognition vocabulary, and so its segmentation methods were specific to the vocabulary, though they could be generalizable. An additional gesture recognizer builds on the concept of $1 to achieve state-of-the-art accuracy using polyline approximation [8]. This recognizer requires few examples and focuses on stroke curvature to determine dominant points for comparison.

Another approach to stroke recognition is to use image processing methods. One multistroke recognizer uses image-based symbol recognition [12]. This system is designed to overcome common challenges with multistrokes and drawing order by merging strokes into an image. They can then be evaluated as one stroke by a distance matching-algorithm. Neural nets have had some success in the field of handwriting recognition as well [4]. One benefit to image – based methods is they simplify stroke comparison. They have limitations in that a multistroke cannot be easily broken down into its components for individual analysis, which may be desirable, and computation time for comparison can be large.

Interactive Machine Learning

SAIGE semi-automates grading with an interactive machine learning (IML) approach to common machine learning algorithms. We choose IML over traditional machine learning

because it allows classification of real problems with limited initial data [15]. This approach also allows users to revise classification during training, which is ideal with responses such as open-ended assessment. To our knowledge IML has not been used in any existing grading approach. However, research into IML provides insights on its potential.

One IML experiment at Carnegie Mellon used human input to learn complex selections of text for editing [16]. Users select text to train the system, and the system suggests additional text to edit. They reduced manual input to average 1.26 examples for correct text selection. This shows that a proper algorithm which can accurately generalize examples for comparison can significantly reduce the amount of manual input required for a task.

At Brigham Young University researchers created the Crayons image classifier, a tool which allows users to iteratively draw over an image to identify objects such as “skin” or “background” [6]. Users select a color and draw a stroke over an area to classify the area. The system uses the highlighted section to generalize over a wider area and identify objects. Added input improves its accuracy. This is useful because it shows that classification by example is possible, and can be reasonably generalized. It is also useful in showing how a non-technical user may reasonably train a classifier using real data, without needing to understand the data representation within the system.

An alternate machine learning method is semi-supervised learning (SSL). SSL is ideal when dealing with large amounts of unlabeled data. SSL can be categorized into data classification or data clustering [10]. SSL classification models have been combined with decision tree and nearest neighbor classifiers with some success [13, 19, 20, 22], in a traditional machine learning setting. A limitation of SSL classification methods is their inherent assumptions about how the data are grouped [22], which may or may not apply to a real set of

graded answers. Future work could include SSL adaptations such as co-training, self-training, or generative models for classification improvement after the limitations have been determined.

Summary

Prior attempts to improve grading focus on reducing grading time using recyclable feedback, text clustering, and automated grading. No grading approaches exist which use interactive machine learning or digital ink stroke recognition. Research into interactive machine learning shows it is possible to learn from few examples and to incrementally train a classifier. Stroke recognition is possible with a variety of feature sets.

5. Choosing the Best Features

Feature selection is an important step for improving classification accuracy and time. For SAIGE, feature selection is also critical for keeping the number of examples low. Learning with a large number of features requires more training examples than with a small number of features, according to the curse of dimensionality [2]. On the other hand, using too few features creates confusion and increased errors due to missing information. It was necessary to choose features meaningful to the classification task and to give them proper weight to optimize learning. We ran a series of experiments designed to explore strokes and determine if there is a best feature set for identifying strokes.

Initial Feature Exploration

We set out to explore features to discover an optimal set for minimizing training examples and errors. We did this by incrementally training on a small set of data and analyzing the results. These experimental results are not particularly formal, but provided useful guidelines for our final feature selection.

Our first step was to determine if there are specific strokes which are common for responding to a question. Using PowerPoint to imitate the SAIGE interface, we created a set of 20 questions which were intended to have depth and variety and could be answered with digital ink. These questions are all adapted from college course quizzes. We then responded to each question with what we considered a probable standard response. A sample of these is shown in Figure 5.1, with responses in green.

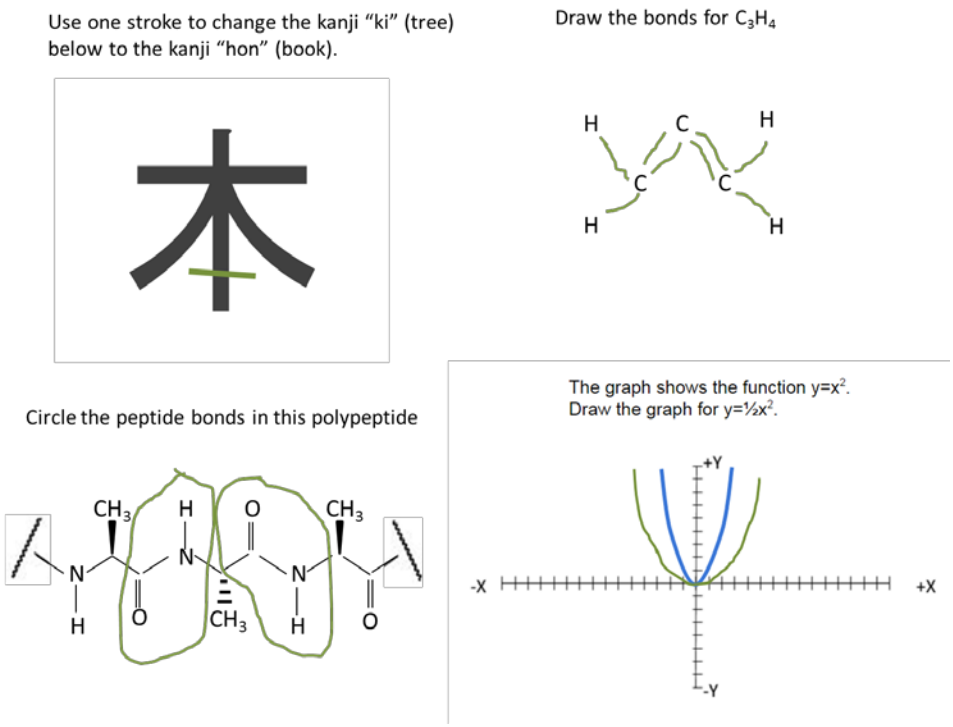


Figure 5.1: Types of digital ink responses

After reviewing the probable responses to each question we concluded that most responses would fall into one of three categories: a line, a circle, or a "path" stroke such as graphs and other complex lines.

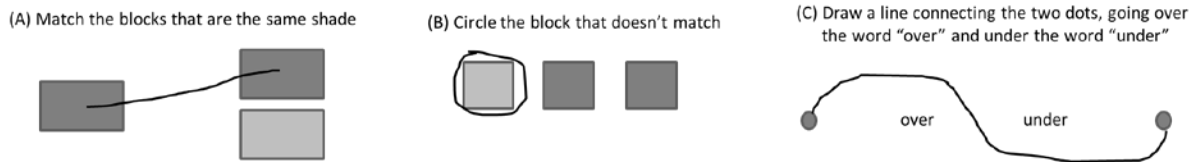


Figure 5.2: Simple Questions for Feature Exploration

We created a test to identify ideal features for the strokes, emphasizing the differences between the three stroke categories. For this test we used SAIGE to create three new questions, one for each stroke type. We responded to the questions with 20 similar strokes. Figure 5.2 shows the questions and example answers we used. Response (A) has a single straight-line stroke, response

(B) has a single circle stroke, and response (C) has a single path stroke. This dataset provided a controlled environment to focus on individual strokes and their attributes without additional clutter or confusion.

We hand-selected small feature sets which focused on specific stroke attributes. We based our feature set on the $1 Recognizer features [23] which use N points along a line. We chose this initial feature set because it has high accuracy for a wide variety of shapes. We included bounding box and endpoints as additional interesting features because intuition stated that these could be optimal features for certain strokes. We tested the following feature combinations:

- n-points only $\{x_1, y_1, x_2, y_2 \ldots x_n, y_n\}$ using n=5
- endpoints only $\{x_{start}, y_{start}, x_{end}, y_{end}\}$
- bounding box only $\{x_{left}, y_{top}, x_{right}, y_{bottom}\}$

We chose to use n-points where n=5 because that generates 10 features and preserves some information about stroke shape. We kept n small to reduce the effects of the curse of dimensionality. We chose to use only endpoints because we felt those would be particularly interesting for identifying lines. We chose to use only bounding box because it appeared to provide the most interesting features for identifying circles.

We then tested combinations of the feature sets on our three simple questions by semi-automatically grading them with SAIGE using a version space classifier. This informal nature of this test allowed us to select an arbitrarily large generalization range for quickly learning responses. Before each test we randomized response order because we expected training order could affect the number of examples required. For each response, SAIGE generated a feature set for the stroke. If the stroke was recognizable within a generalization range, it was autograded.

Otherwise it was added as training data. We tracked the number of examples required for each question and averaged the results over 11 tests.

Figure 5.3 contains the results for the initial feature test. Each result is the average number of handgraded responses required to complete grading 20 identical responses, averaged across 11 tests.

	Average Number of Examples Required (out of 20)		
	Endpoints-only (EO) (4 features)	Bounding-box (BBO) (4 features)	n-points (10 features)
Response A (line)	6.09	6.09	8.27
Response B (circle)	5.36	4.9	7.36
Response C (path)	5.54	6.18	8.54

Figure 5.3: Results of Feature Selection Test

Analysis of Results

A first glance at the results shows that by the curse of dimensionality, using a reduced number of features reduces the number of examples required. It also verifies that we can adequately learn a single stroke in fewer than ten examples using a large amount of generalization. Every response was identical, with only one stroke per response, so this test did not directly address how features affect misclassification. However, an analysis of the chosen features reveals insight about an ideal feature set.

Response A required an average 6.09 examples using endpoint-only (EO) and bounding box-only (BBO), 2.17 examples fewer than n-points. The numbers indicate that EO and BBO features are equivalent for straight lines, but there is a clear advantage using EO for lines. Consider the case in Figure 5.4. The first response is correct, and the bounding box is shown in red and endpoints in blue. The second response is incorrect, again with the bounding box and

endpoints marked. Although with EO the strokes are distinct, with BBO they appear identical, and misclassification during autograding would occur. We conclude that EO is the optimal feature set for lines.

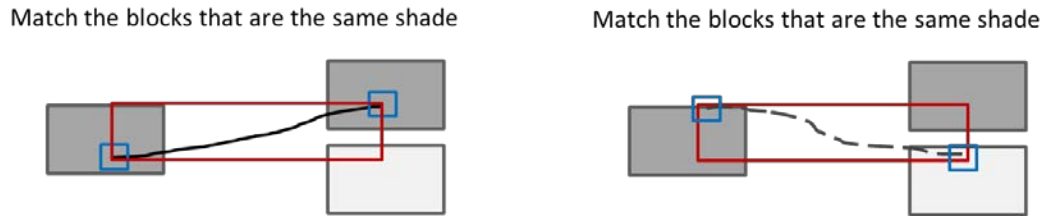


Figure 5.4: Comparing line features

Response B required 4.9 examples with BBO, lower than EO or n-points. Further analysis reveals an additional advantage for BBO over EO or n-points. The case in Figure 5.5 illustrates this advantage. The first two responses are similar, but one was drawn with endpoints at the top, and the other was drawn with endpoints on the side of the circle. The third response was drawn with the endpoints on the left side of the circle. If all responses were similar to the first two responses, more examples would be required to learn the response. Similar problems occur with n-points. In addition, when responses resemble the last two responses, then misclassification errors occur with EO. We thus conclude BBO is the ideal feature set for a circling problem.

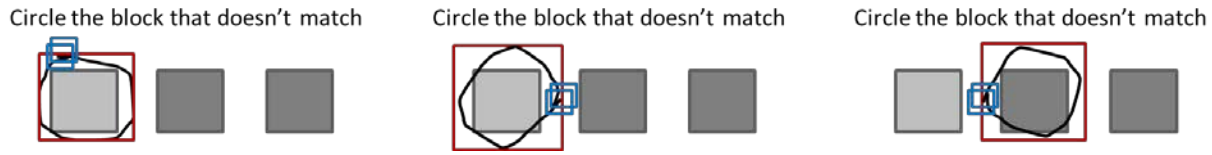


Figure 5.5: Comparing circle features

Response C, a path problem, also required the fewest examples using EO and BBO. However, these features are not sufficient for complex strokes. Consider the example in Figure 5.6. These dissimilar strokes have identical endpoints and bounding boxes. The key information

about the stroke is its shape, and using EO or BBO loses that necessary information, making misclassification likely. Thus n-points is the best option for path strokes despite the training cost.

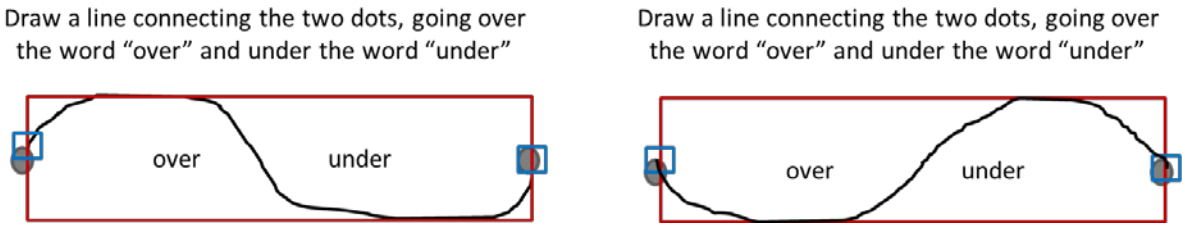


Figure 5.6: Comparing path features

Hit Features

We also investigated the background objects a stroke touches as sources of meaningful information. For example, in Figure 5.4, knowing which rectangles the stroke touches is more meaningful than knowing the absolute location of the stroke. In this example each endpoint of the line hits a rectangle. SAIGE can identify which point touches which rectangle, and thus learn about the context of the stroke within the problem. We tested a few combinations of hit features using different points on the stroke, including corners of bounding boxes, endpoints of the stroke, and center of the stroke. This gave us interesting context information about the stroke, which we expected could replace or supplement existing features and allow some generalization without additional work. We added hit points to our features and were surprised to see no improvement, so we tested hit points without other features to determine why. We used real data from two questions, with approximately 300 responses each, and compared results of using bounding box hit features to using absolute location of endpoints. The two questions with their respective test results are shown in Figure 5.7 and Figure 5.8.

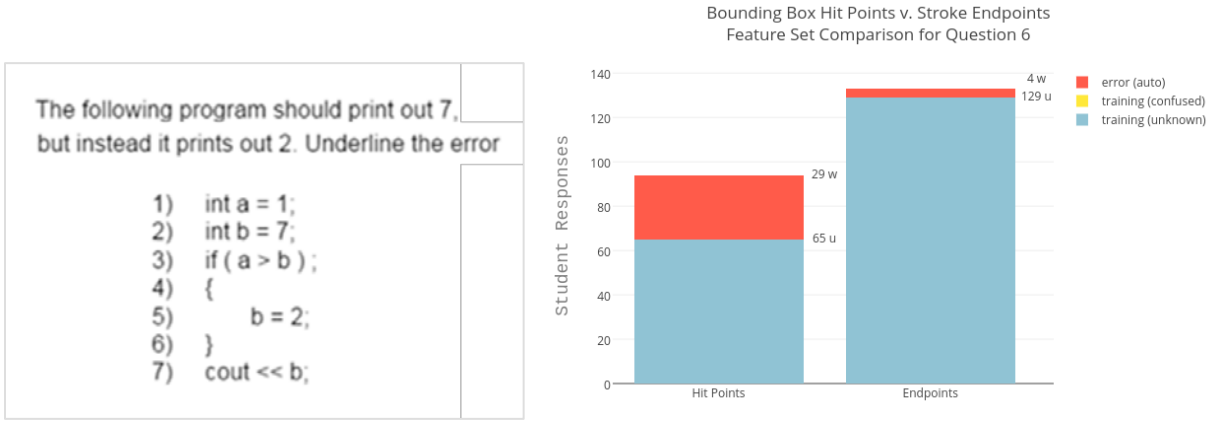


Figure 5.7: Question 6, errors and training from using hit features or endpoint features

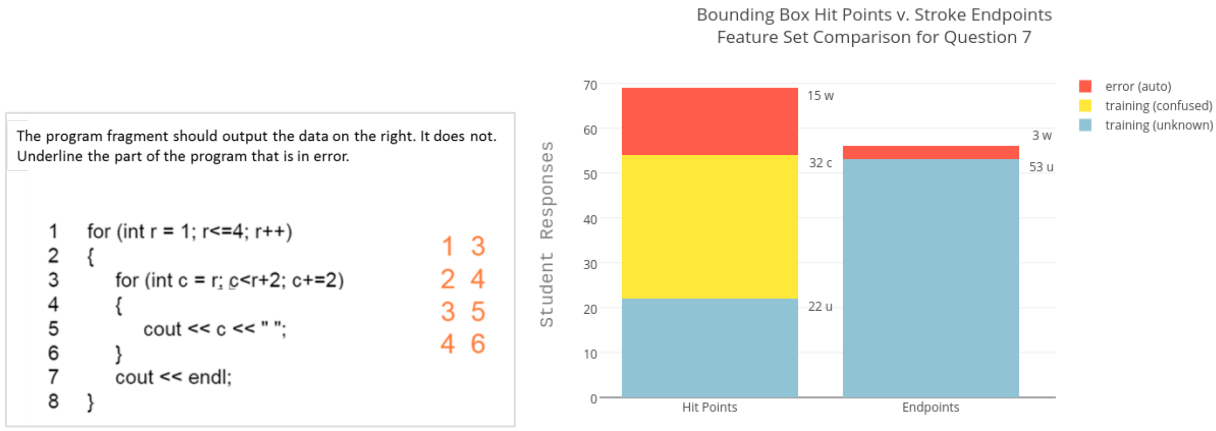


Figure 5.8: Question 7, confusion, errors, and training from using hit features vs endpoint features

The first graph in Figure 5.7 shows that using hit points instead of endpoint location generalized more, but also showed a surprising increase in number of errors. The second graph, Figure 5.8, showed a comparable amount of training overall, but with an increase of confusion during classification. Confusion occurs when multiple classes seem correct, so SAIGE requests help to learn the difference.

Analysis of Results

We reviewed the most commonly confused question responses from the test to determine the reason for confusion. Both questions relied heavily on text background objects, and each stroke

was intended to mark a location on a text box. The two most common responses to one question are shown in Figure 5.9. The strokes fall under the same line of text, with one stroke spanning the whole line and the other spanning a short section of the line. We reviewed the hit features and feedback assigned to each stroke and discovered the classification problems were caused by whitespace characters and character bounding boxes.

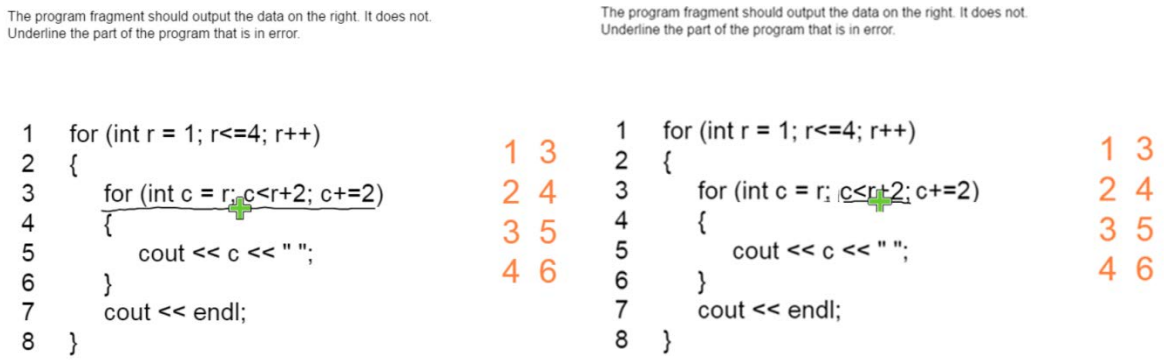


Figure 5.9: Hit test data most common responses

The two sample responses in Figure 5.10 illustrate the problem caused by whitespace and character bounds. The first example highlights the character bounds for each character. The top and bottom of the character bounds extend beyond the top and bottom of each letter. In addition, whitespace is a single large character, not multiple small characters. Each character has an index based on order, and when a stroke touches the character, the index becomes the hit feature for the stroke. These qualities cause confusion for classification. In the first example, the two blue strokes deserve the same feedback, but hit different characters. Especially with a version space classifier, which relies on range of features, this is a problem because the blue lines (characters 44-48 and 52-54) create a feature range that encompasses the red line so it receives the same feedback. In the second example, all the strokes fall within bounds of a single whitespace character. Because every point along the stroke hits the same character index, the hit features for

each stroke are identical although the strokes are different. This causes confusion for any classifier.

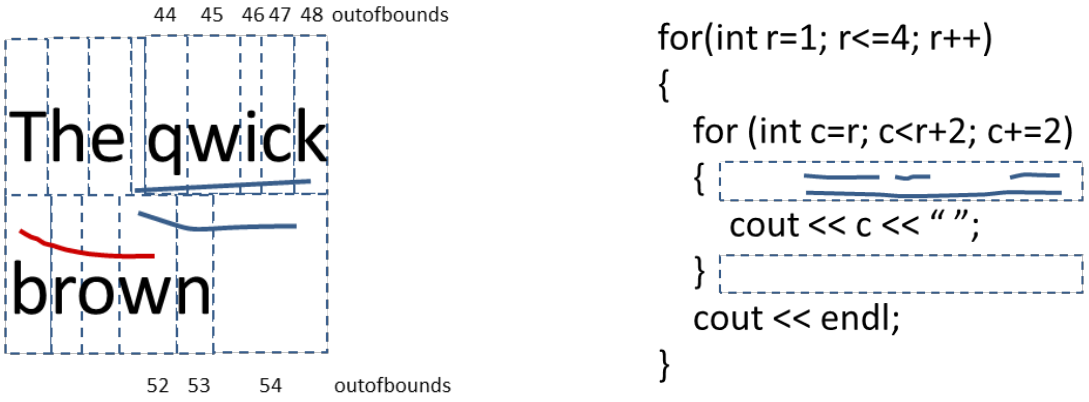


Figure 5.10: Whitespace and character bounds provide challenges using text with hit features.

Conclusions

These experiments led us to conclude that there is not one universal best feature set for minimizing training examples and preventing errors. For line or circle identification, we can reduce the number of training examples required by specializing which features to use. For straight lines, it is sufficient to use only endpoints. For circles, it is sufficient to use bounding box. These features are not sufficient for paths, but if we could distinguish lines and circles from paths, we could reduce the total number of examples required for grading a quiz. Hit points, unless combined with other features, create confusion and error, and do not provide enough value to be useful. This led us to implement an algorithmic stroke partition to differentiate stroke types. Using this partition we would be able to select the minimal features for stroke classification.

Algorithmically Partitioning Strokes

For each stroke in a response, we algorithmically identify the stroke type using the following three shape definitions:

- Line – endpoints should be relatively far apart and endpoint distance will approximate the length of the stroke
- Circle – endpoint location can vary widely but will be close together. Length has to be much longer than the distance between endpoints.
- Path – The distance between endpoints can vary widely, length will be much longer than the distance between endpoints

We normalized stroke size (bounds) for determining stroke type to guarantee short straight strokes were not classified as circles or paths.

Feature comparison requires that features must be consistent. It is not reasonable to consider comparing bounding box to endpoints. Thus we separate examples using three classifiers, one for each stroke type. We check the stroke type and add a training example or grade an unidentified stroke using the corresponding classifier. This means each classifier has fewer examples to generalize on, which means the total number of examples may increase if there is a wide variety of stroke types in a set of responses.

Testing the Partition

We tested the partition on a new small set of controlled data which was similar to the feature exploration dataset. There were three questions in this dataset, each with 15 similar correct responses and 10 similar incorrect responses, shown in Figure 5.11.

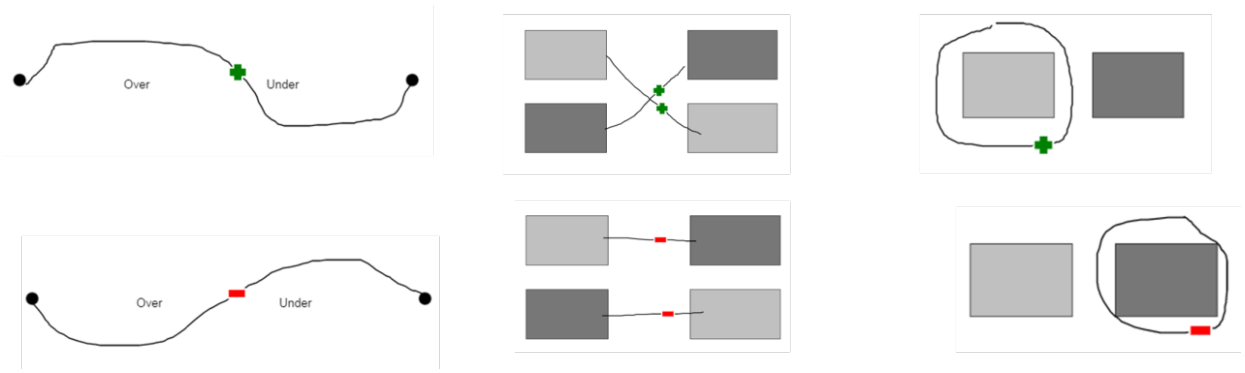


Figure 5.11: Simple questions for partition test

We used a nearest neighbor classifier with a large generalization distance of 100 pixels for the test. Figure 5.12 below shows the final results, averaged over five tests. Partitioning lines and circles reduced their average number of required examples by 4 and 7.8 respectively, and had no negative impact on number of errors.

	Best Average			Worst Average		
	Features	Example Responses	Errors	Features	Example Responses	Errors
Line	Endpoints	3.9	0	n-points	7.9	0
Circle	Bounding box	3.3	0	n-points	12.1	0
Path	n-points	5.1	0	Endpoints	2	9.3

Figure 5.12: Results from using differentiated features for classification

Evaluation of Success

Since there are three simple types of stroke which are common responses, differentiating between strokes to select features is one way to minimize the total number of examples and errors during grading. Choosing the right features alone does not perfectly meet our metrics, but reducing the number of features when possible and selecting a few meaningful features makes a big difference.

6. Perversity of Strokes

Stroke perversity occurs in many forms. Perversity can occur from human error and response ambiguity or from the inherent fact that strokes with the same meaning can take multiple forms. In any environment that is not perfectly controlled, we find students do not follow directions, they misunderstand the interface, and their answers will overlap and create grading ambiguity. In addition, even strokes with the same meaning take a variety of forms. This poses a challenge to grading because more perverse data means more training examples and more errors. We had to discover if SAIGE could handle perversities automatically to minimize the effect on grading.

Discovery of Stroke Perversities

Two datasets revealed various types of stroke perversity. In this section we discuss the two datasets and each perversity they revealed. The first dataset used a basic graphing problem to generate a set of responses. The task was to draw a graph to represent $y=\frac{1}{2}x^2$.

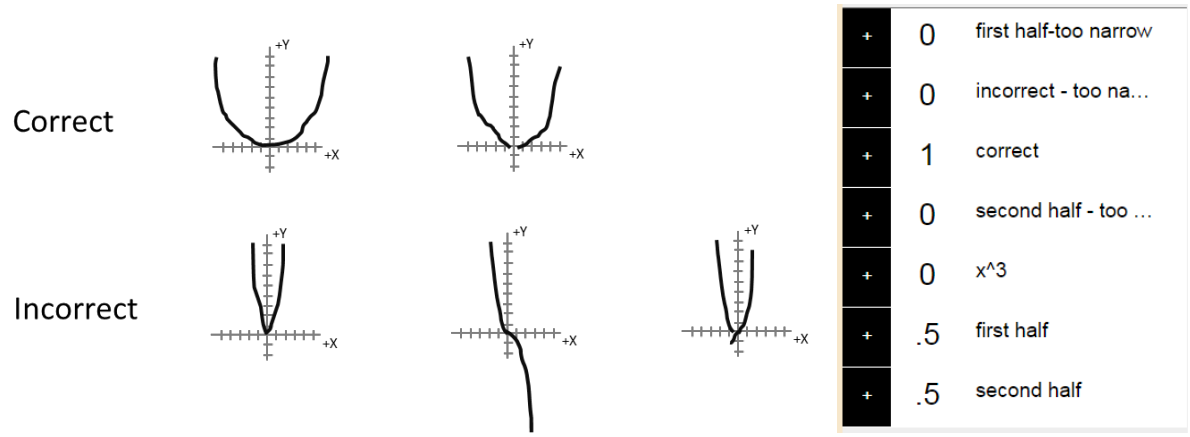
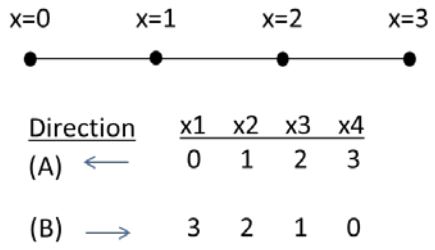


Figure 6.1: Example Responses and Feedback for Small Dataset

We created 25 responses for each question in a controlled environment. We drew 15 responses to correctly represent $y=\frac{1}{2}x^2$. We also drew ten "incorrect" responses which included seven graphs of $y=2x^2$ and three graphs of $y=x^3$. Figure 6.1 shows a sample of the responses. The dataset included some variation to simulate expected variety from a set of real answers. Seven feedback items were initially required to adequately label the data. Creating this dataset introduced us to two types of stroke perversity: multistrokes and stroke direction.

Stroke Direction

Considering the strokes' drawing direction illustrates an important insight for reducing training examples and preventing errors. A stroke is a vector of points, which are drawn in a certain order. Visually identical strokes can have very different features if drawn in opposite directions, causing negative affects during training. Using a k-nearest neighbor classifier, actual stroke distance will not match visual stroke distance, requiring up to twice as many examples to train. Using a version space classifier, the actual ranges in a hypothesis will not accurately reflect the visual feature range.



K-Nearest Neighbor	Version Space
Visual Distance(A, B) $= \sqrt{(0-0)^2 + (1-1)^2 + (2-2)^2 + (3-3)^2}$ $= 0$ Actual Distance(A, B) $= \sqrt{(0-3)^2 + (1-2)^2 + (2-1)^2 + (3-0)^2}$ $= \sqrt{20}$	Visual Hypotheses = Feature Ranges(A, B) = [0, 0] [1, 1] [2, 2] [3, 3] Actual Hypotheses = Feature Ranges(A, B) = [0, 3] [1, 2] [1, 2] [0, 3]

Figure 6.2: Effects caused by training with visually identical but directionally different strokes

Figure 6.2 illustrates the problems stroke direction causes. A one-dimensional stroke can be drawn in two different directions, which are two distinct vectors. The example uses each x-value as a feature, similar to our n-points feature set. Using k-nearest neighbor with Euclidean distance for comparison, the true distance between vectors A and B is much larger than the

visual distance, and would fall outside of a reasonable maximum distance stopping criteria. In order to effectively learn strokes in opposite directions SAIGE would require examples of each stroke direction, doubling the number of examples. A version space hypothesis built using vector A would not fit vector B without expanding the feature ranges. In this case, doing so creates an overlapping range of possible values for each feature, and the true shape of the stroke is not preserved. Thus SAIGE must resolve stroke direction to recognize the same stroke drawn in the two opposite directions.

Multistrokes

While responding to the question, it was natural to draw the stroke in two halves rather than one whole. The teacher can individually label each half and the classifier would be able to classify the halves. However, there is a problem with autograding the two halves separately. Figure 6.3 illustrates this problem. The response on the right has one half that closely resembles half of the expected parabola (left), and was given the "correct, left half" feedback. The assigned feedback is nonsensical in context because the response as a whole represents a different stroke.

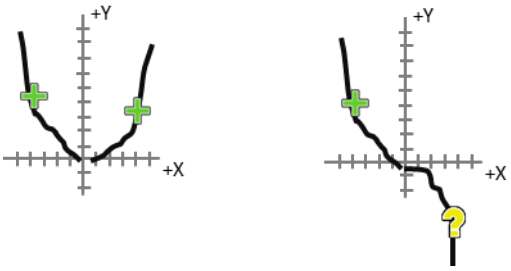


Figure 6.3: Classification errors from split strokes

This intuition was confirmed by running a test on this dataset. Five classification errors occurred due to multistrokes. SAIGE had a need to combine multistrokes to allow one feedback for a whole response.

Meaningless Strokes and Crossout Strokes

The second dataset was real data from 360 students enrolled in a Computer Science course. We discovered perversities while grading semi-automatically and reviewing the results. Figure 6.4 shows a sample of responses. On the left is the response that the majority of the class submitted. On the right is an example of meaningless strokes. The student misunderstood the interface and wrote an explanation of their response in digital ink strokes which SAIGE expects to grade letter by letter. Although sometimes informative to read, these strokes are meaningless to the learning algorithm.

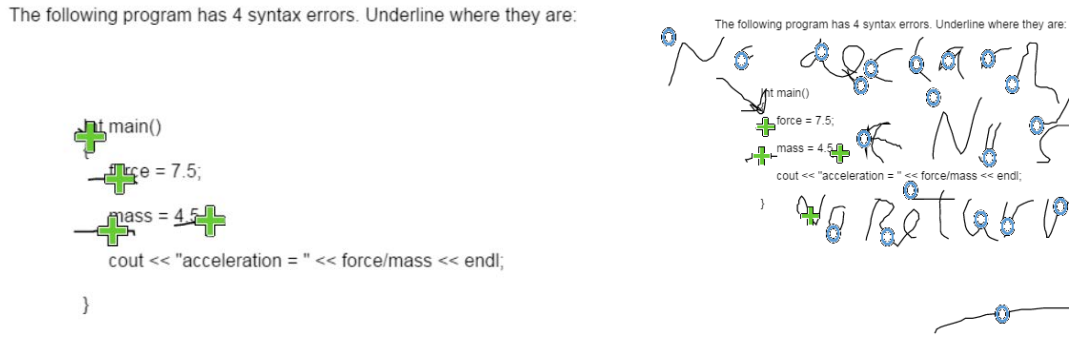


Figure 6.4: Sample Responses

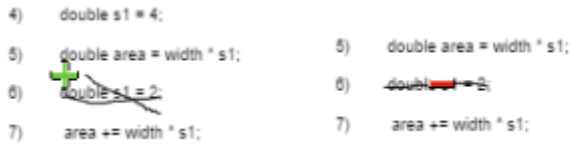


Figure 6.5: Crossout Stroke as Single Response

We also discovered an additional type of split stroke, crossout strokes, shown in Figure 6.5 on the left. These are intersecting strokes which should be treated as a single response, as when drawing an "X." Like multistrokes, crossouts should be treated as a single response, like

the single stroke on the right, but because they are two individual strokes they each receive individual feedback, which poses a challenge to grading if not addressed.

Addressing Stroke Direction

We explored stroke duplication and stroke normalization as approaches to resolving stroke direction. Figure 6.6 shows the process for these approaches. In each case, when SAIGE receives a labeled stroke it preprocessed the example to generate better training data. In the case of stroke duplication, during training SAIGE creates a second stroke from the first by copying its point vector in reverse order. It then adds both strokes to the training data. At classification time, strokes are compared to the augmented training data. Stroke normalization uses a stroke direction analysis during training and classification to determine whether the stroke is drawn in clockwise order. If not drawn in clockwise order, the stroke's point vector is reversed. At training time a single normalized stroke is added, so the training data is normalized rather than augmented, and at classification time normalized strokes are compared to normalized training data.

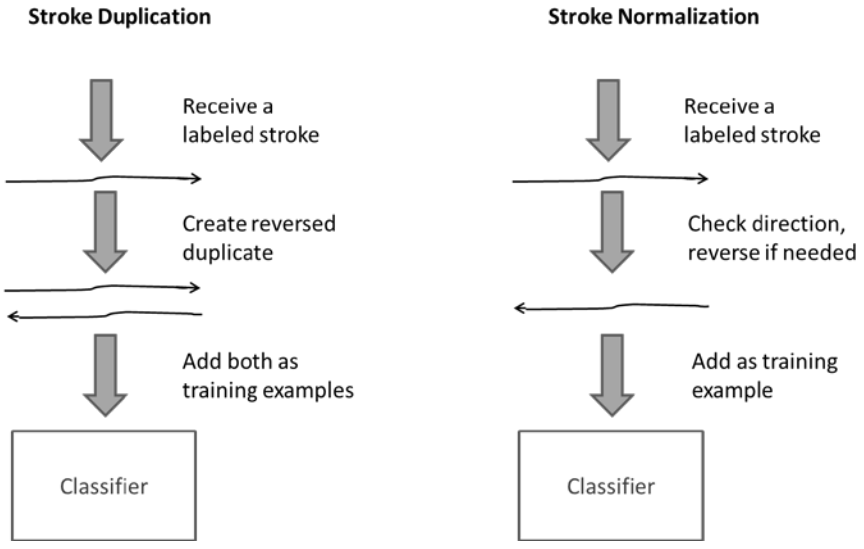


Figure 6.6: Processes for Stroke Duplication and Stroke Normalization

Stroke duplication had the advantage of simplicity and of generating double the number of examples at training time. This method worked for KNN, but created problems for version space.

The version space classifier uses examples to create bins of maximum and minimum values for each feature. It classifies by determining which set of bins will fit the stroke's features. Bins should be distinct for a feature, as illustrated in Figure 6.7 on the left. Stroke normalization restricts the bins to their ideal shape by guaranteeing all strokes are drawn in the same direction. However, adding strokes which have two different directions inaccurately widens the range of each bin, increasing the probability of misclassification. This scenario is illustrated in Figure 6.7 on the right.

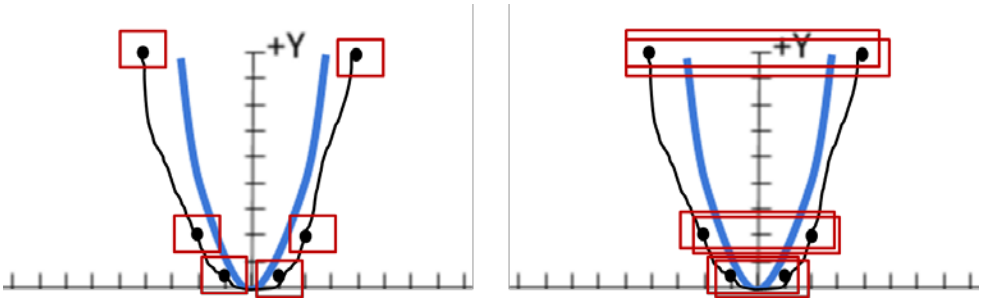


Figure 6.7: version space feature bins

Addressing Multistrokes

We implemented a stroke combination algorithm to merge multistrokes automatically using a 2Nx2N matrix to determine which stroke endpoints touched. We used a maximum distance function to compare the four possible endpoint matches (start-start, start-end, end-start, end-end) for every stroke pair in the response and filled the matrix. The resulting matrix is a symmetric representation of where strokes connect. Figure 6.8 contains an example matrix, with a "1" for a match, a "0" for no match, and "X" along the diagonal. SAIGE then parses the upper triangular part of the matrix to remove junctions of three or more strokes, which cannot be sensibly

combined, and extracts the remaining stroke pairs. Then SAIGE merges each stroke pair into a single unified stroke and replaces the individual strokes with the unified stroke. This takes $O(N^2)$ time in the number of strokes, but most responses contained fewer than five strokes so the impact was minimal.

	Stroke 1 start	Stroke 1 end	Stroke 2 start	Stroke 2 end
Stroke 1 start	X	X	0	0
Stroke 1 end	X	X	1	0
Stroke 2 start	0	1	X	X
Stroke 2 end	0	0	X	X

Figure 6.8: Example Endpoint Matrix

We ran an experiment to measure improvement using stroke normalization with multistroke combination. The test used 4 questions with 25 responses each (100 total responses). Each question contained one or more strokes to make a total of 172 strokes to grade. Before the modifications, SAIGE required 27 example strokes (20 responses). After including multistrokes and normalization SAIGE required 19 strokes (16 responses), 2/3 of the prior workload.

Addressing Crossout Strokes

We implemented a preliminary solution for merging and classifying crossout strokes using a matrix to determine stroke intersection. To reduce computation time we used an approximate intersection algorithm to determine which strokes intersect. The steps are shown in Figure 6.9. First we checked the stroke bounding boxes. If there was no overlap, they could not intersect. If there was overlap we checked for intersection between the strokes by determining if the lines between endpoints intersected. Once intersection was determined we could differentiate between single lines (line, circle, and path) and intersecting crossout strokes.

Figure 6.9: Steps to define a crossout stroke

We decided to ignore crossout strokes in autograding since comparing more than one stroke becomes complex to analyze. Ignoring crossout strokes from the mix of answers reduced the number of required examples by up to 4.69% with no impact on errors, which indicates that crossout strokes were prevalent enough in our data to bear further consideration in the future.

Addressing Meaningless Strokes

Meaningless strokes are best determined using human judgment, because in our implementation humans have context which SAIGE lacks. As such, we created an "ignore" option to auto label any ungraded strokes with an "ignore" flag by pressing a button at the end of grading a question. This flag was a hardcoded label which SAIGE would recognize, and we instructed SAIGE not to learn ignored strokes, and not to use the ignore label in classification. We chose to do this because meaningless strokes come in such variety that SAIGE could not effectively learn them with this implementation, particularly using a KNN classifier. Although meaningless strokes were not learnable with this implementation, labeling all unlabeled strokes at once with this method reduced the amount of individual labeling a teacher had to do, so there was a workload reduction of up to 43 seconds of grading for each of these responses. Future work would include considering a UI change to highlight a gradable area and only consider strokes in that region.

Evaluation of Success

We discovered four perversities when using stroke data for grading: stroke direction, split strokes such as multistrokes and crossouts, and meaningless strokes. Each perversity contributes to an increase in number of examples, number of errors, or both during training and classification. We normalized stroke direction to ensure all strokes are compared in the same direction and merged multistrokes into single strokes. Normalized stroke direction and merged multistrokes together reduced training from 27 to 19 strokes on a dataset with 172 strokes, with errors reduced from five to zero responses in a dataset of 25 responses. We identified and removed crossouts from the data to determine their prevalence, and determined that they made up 4.69% of the strokes, enough to address in future work. We created an "ignore" option to allow teachers to skip grading meaningless strokes. SAIGE could not learn these strokes, but the ignore option decreased time spent grading by up to 43 seconds per question. Future work would require a way to work more effectively with meaningless strokes.

The ways we addressed stroke perversity improved SAIGE's performance with regards to time, errors, and examples. Additional work is required to completely resolve the problem of crossouts and meaningless strokes, which may require significant changes to our current interface.

7. Selecting a Classifier

Incremental training poses a unique challenge compared to standard machine learning. Standard machine learning algorithms are trained on all possible training data at the beginning of the classification process, and their goal is to make the best possible decision based on that data. SAIGE also makes the best classification decision possible based on its training data, but it begins classifying with almost no initial training data. Since our metrics demand that SAIGE stop asking for training as soon as possible to maximize automation, SAIGE's training data is limited to10-15 training examples rather than the 1000-1500 examples a standard algorithm would expect. To achieve high accuracy and minimal training, we required a classifier that could both generalize well on limited training data and recognize when to request additional training using stopping criteria.

Initial Classifier Selection

We initially considered the following classifiers: naïve Bayes, decision tree, k-nearest neighbor (KNN), version space (VS), deep neural networks (DNN), and support vector machines (SVM). Classifiers such as DNN and SVM will often outperform more basic machine learning models such as KNN in terms of accuracy. This makes these classifiers attractive for many machine learning problems. However, DNNs require a large amount of training time as well as fine-tuning of parameters which make them unattractive in the context of user-friendly, low barrier of entry interactive machine learning. SVMs are simple and can be configured to provide some measure of certainty for stopping criteria, but SVMs are binary classifiers. Although there are implementations of SVMs which perform multi-class classification, we applied the principle of Occam's razor to conclude that using a naturally non-binary classifier could be more desirable. This left us with four options to explore. We then narrowed our selection using four

mandatory criteria based on SAIGE's grading paradigm. The first criterion was ability to learn incrementally. Without incremental learning the classifier is limited to classification based only on initial training data. Our paradigm was to incrementally add training data when necessary, and SAIGE's classifier had to be able to update its learning with incremental addition of new training data. Each classifier can learn incrementally. KNN requires no recomputation with additional training data. VS may require re-computing bounds for a hypothesis, naïve Bayes requires re-computing probabilities and decision trees require rebuilding the tree when additional training is added.

The next criterion was ability to generalize with stopping criteria. SAIGE was to grade as much as possible with few examples by generalizing, and ask for additional training when a stroke is unrecognizable using a stopping distance function. Without generalization, training is unwieldy because a larger number of examples is required to recognize strokes. Generalization allows us to reduce the amount of training. Our grading paradigm also requires SAIGE's classifier to include some natural stopping criteria to inform SAIGE that a stroke is unrecognizable. This is necessary so SAIGE will avoid overgeneralization and ask for training when needed. Naïve Bayes, KNN, and VS each have reasonable inherent stopping criteria for generalization. Naïve Bayes can use probability of confidence, KNN can use a maximal distance function, and version space can swell a hypothesis' bounds using some factor. A standard decision tree does not have obvious stopping criteria, although its variations could provide some probability measure which could be used.

The third criteria was that training and classification be $O(n^2)$ or less. A good user experience includes a rapid response, and response time greater than $O(n^2)$ is too slow for

interactively training and using a machine learning program. A basic decision tree, KNN, and VS are all $O(n^2)$ or less for training and classification.

The final criterion is ability to work well with few examples. We expect class size to be between 20 to 400 students, which means we begin with a small number of responses to train with. Our goal is to limit training to 10% of all responses in order to keep teacher workload small. This restricts our training data to 10-40 responses. SAIGE's classifier must be able to learn effectively enough to accurately classify with this small amount of training. Decision trees, KNN, and VS each will work reasonably well with few examples. The low number of training examples would not provide enough information to create an accurate probabilistic model using naïve Bayes.

	Incremental Learning	Generalize with Stopping Criteria	Training/Classification $\leq O(n^2)$	Works with few examples
Naïve Bayes	With recalculation	Yes – probability	Yes	No
Decision Tree	With recalculation	Maybe - probability	Yes	Yes
KNN	Yes	Yes – distance	Yes	Yes
VS	With recalculation	Yes – bounds	Yes	Yes

Figure 7.1: Strengths and Weaknesses of Four Algorithms

Figure 7.1 shows a summary of how well each classifier meets the four criteria. We chose not to pursue naïve Bayes and decision tree based on their disadvantages, since the number of examples, training time, and stopping criteria are critical for SAIGE's success. KNN and VS had additional advantages. KNN has been used successfully for recognizing single-stroke characters with the $1 Recognizer, and VS is well suited to discriminate between features. There was not an obvious win between KNN and VS at this stage, so we implemented several variations for both and compared their performance. Since this is a specialized problem, for greater flexibility we

chose not to use existing libraries. Further discussion of the parameter specifications are discussed in the following sections.

We used our large dataset to determine the limitations of each classifier. This dataset contained eight questions with between 265 and 361 responses each. We selected one set of features to look at for this comparison to determine how the classifiers compare without confounding factors. We used the differentiated line, circle, and path feature set, which small tests had shown had the highest accuracy with few examples. During each test we identified where classifier confusion and errors occurred and counted the number of examples required. We chose not to run any standard tests with a training-testing split and cross-validation, due to the fact that what matters most in this setting is how it performs interactively. Rather, our tests incrementally add training examples until all the data has been classified, simulating a human experience.

K-Nearest Neighbor

KNN uses a distance function to determine which k examples a new data instance is nearest to, and assigns it a label according to the neighbors. We implemented three variations to KNN to adapt it to incremental training and grading. Since our number of examples was small, we only tested k=[1, 3]. We calculated nearest neighbor using Euclidean distance of features, with a distance of 200 or 0 for nominal features. We did not normalize features because doing so would remove underlying data about location, which matters in this problem. Stopping criteria used a maximum distance function and voting about neighbors' classes as its measures of uncertainty to avoid overgeneralization and misclassification.

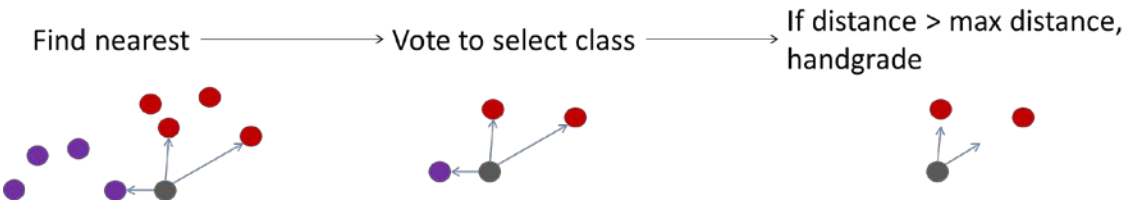


Figure 7.2: K-Nearest Neighbor Implementation

When SAIGE finds the k nearest neighbors, it then must decide if the stroke is recognizable in comparison based on a measure of uncertainty. Using a unanimous voting approach, if all the nearest neighbors do not have the same class the stroke is considered unrecognizable. Otherwise SAIGE checks the stroke's distance to each neighbor against the maximum distance (MD). With a majority vote approach, SAIGE selects the majority class and compares the stroke to the majority examples. If any distance is greater than MD, the stroke is considered unrecognizable. We tried two approaches to generating MD. The first variation was a naïve approach using set pixel distance. Since stroke length is variable, we hypothesized that a best maximal difference between strokes would also be variable, and a set pixel distance would be non-optimal. This led us to implement an adaptable maximum distance (AMD) based on training examples. In this approach we calculate the maximal distance between the examples for each class in the classifier during training. When we have found the k nearest neighbors, we look at their classes and use the class-specific AMD as the stopping criteria.

We began by testing KNN using k=1 and max distance of 100px. We began with these parameters because it was a simple configuration which would still be informative about possible optimizations. Using MD=100px was a moderate generalization based on the size of the drawing canvas (1200px wide), and k=1 is the simplest version of KNN. In this and every test, we count the number of errors, the number of training examples, and the number of correctly autograded responses. In the graphs we differentiate training examples by examples that are unknown due to

classifier confusion and those that are unknown due to distance. Classifier confusion for KNN occurs when all neighbors must have the same class to assign a grade (a unanimous vote), but the neighbors do not all match. In this case SAIGE asks for manual grading. We differentiate in order to understand the reason for changes to the amount of training, and to understand where errors are more likely. With k=1, training due to confusion does not occur because there is only one neighbor.

Figure 7.3 shows the results of the initial test. As the figure shows, each question has slightly different results but overall the amount of training fell between 3% and 11%. Only two questions required more than our desired goal of 10% training. The number of errors has a wider range, between 0.9% and 16%. Five of eight questions had errors greater than our goal of 2%.

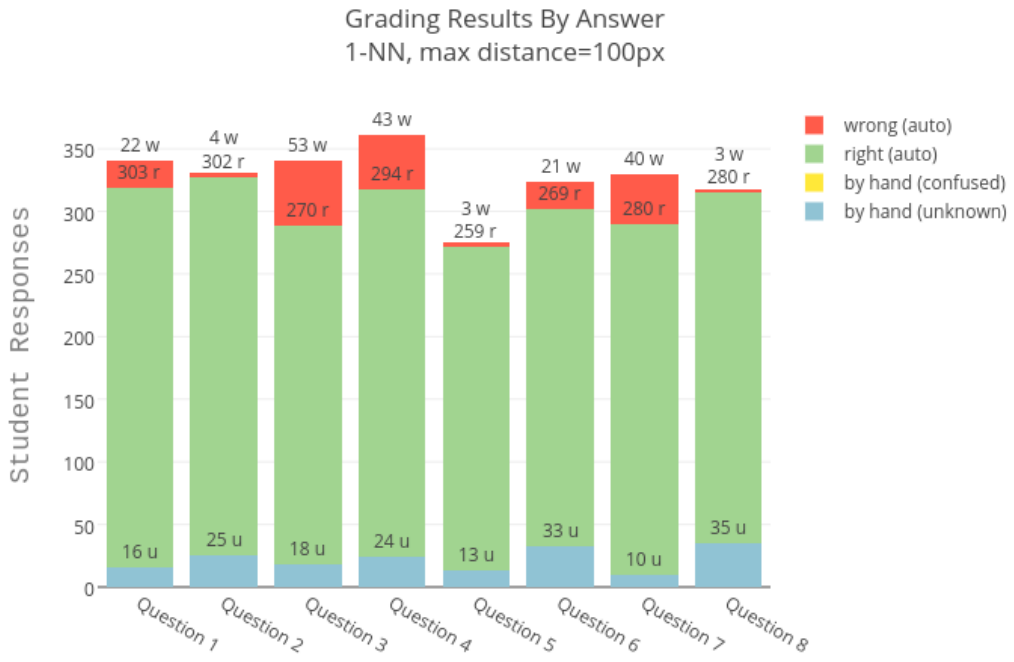


Figure 7.3: Initial results for KNN for eight questions

We ran subsequent tests to determine how much SAIGE could improve using different values of k and different stopping criteria. Six tests are shown in Figure 7.4, with the total responses with errors, responses used as training examples, and responses autograded correctly for Question 4, given as percentages. In the initial test this question had a high number of errors and high number of examples compared to the other questions. As the table shows, there was a tradeoff between errors and examples based on which stopping criteria were used.

	Total Errors	Total Examples	Autograded correctly
K=1 MD= 100px	12%	7%	81%
K=1 MD=50px	9%	13%	78%
K=1 AMD	9%	11%	80%
K=3 MD=100px, majority	17%	69%	76%
K=3 MD=100px, unanimous	5%	18%	77%
K=3 AMD, unanimous	2%	20%	78%

Figure 7.4: KNN Adaptation Results for Question 4 (361 responses)

Reducing set distance decreased the number of errors, but increased the number of examples. Of note is that the errors and training for k=3 MD=100px majority vote and k=3 MD=100px unanimous vote are inverses. This effect is due to classifier confusion. Using majority vote, there is a greater probability of misclassification than if all neighbors must match. Conversely, with a unanimous vote, there will be a greater number of training examples and fewer errors because SAIGE asks for help when the results are uncertain.

Classifier confusion occurred most frequently when two strokes of different classes were contextually different, but physically close, creating ambiguity in feature space. For instance, Figure 7.5 shows a possible training scenario. The teacher grades the first and second responses, assigning feedback “capitalize int” to A and C, and “missing int before force” to B and D. SAIGE then attempts to autograde the third response, which contains stroke E. The teacher may believe stroke E should receive the same feedback as A and C, because all three strokes fall

above the word “force.” However, stroke E is very slightly closer in feature space to D than A, which means using a k=1 classifier, SAIGE will label it with the feedback D received, and so causes a grading error. Responses such as these with strokes close together cause the context of strokes to become much more important than absolute position, which our feature set was based on. Using a unanimous or voting classifier would improve the probability of correct classification, but the basic challenge remains the same. SAIGE with KNN had no way to interpret context such as stroke position or background information to assist with classification, and that was a leading cause of confusion and errors.

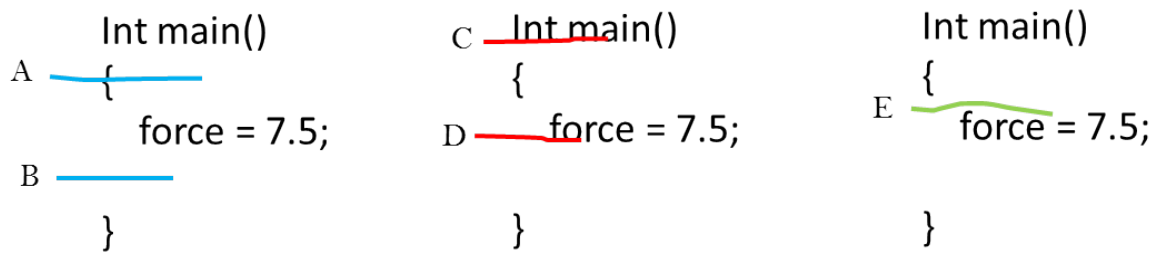


Figure 7.5: Using Nearest Neighbor with Relative Positions

The following graphs show the results of the two best KNN configurations, one which had the fewest errors for all questions and one which had the fewest training examples. Due to the tradeoff between overgeneralization and undergeneralization, there was not one best KNN configuration for both errors and examples.

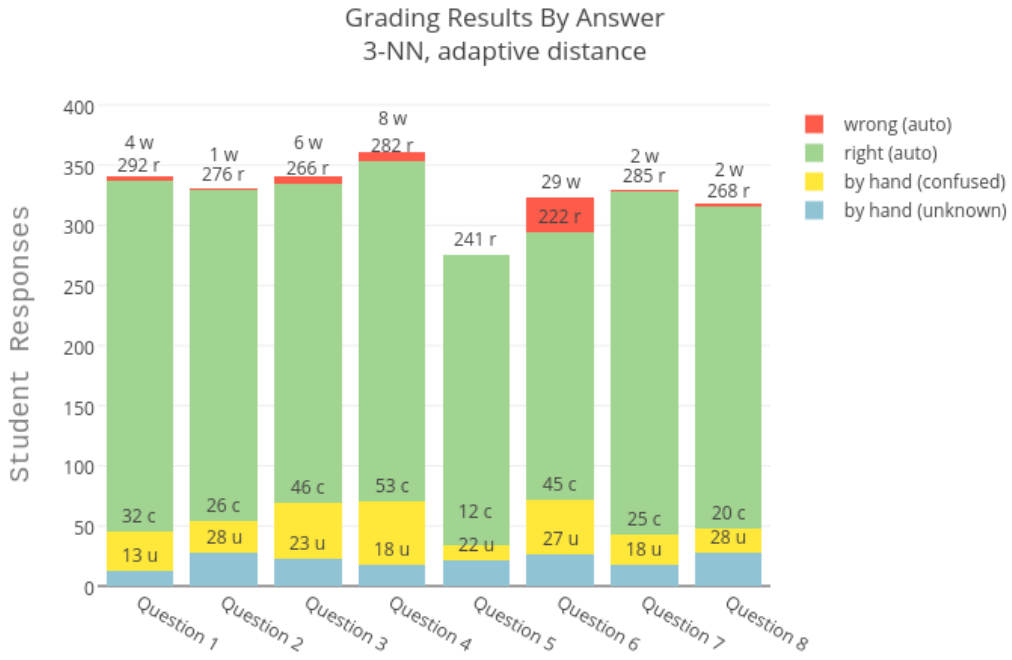


Figure 7.6: KNN configuration with fewest errors

Figure 7.6 shows the results for all questions using k=3 AMD with unanimous vote. This combination achieved the lowest errors for all questions. Seven of the eight questions meet our goal of ≤2% error. Question 6 had 9% error, much of which was caused by stroke perversity. Total training examples fell between 12 and 22%, greater than our goal of 10%. Most training was due to classifier confusion during voting.

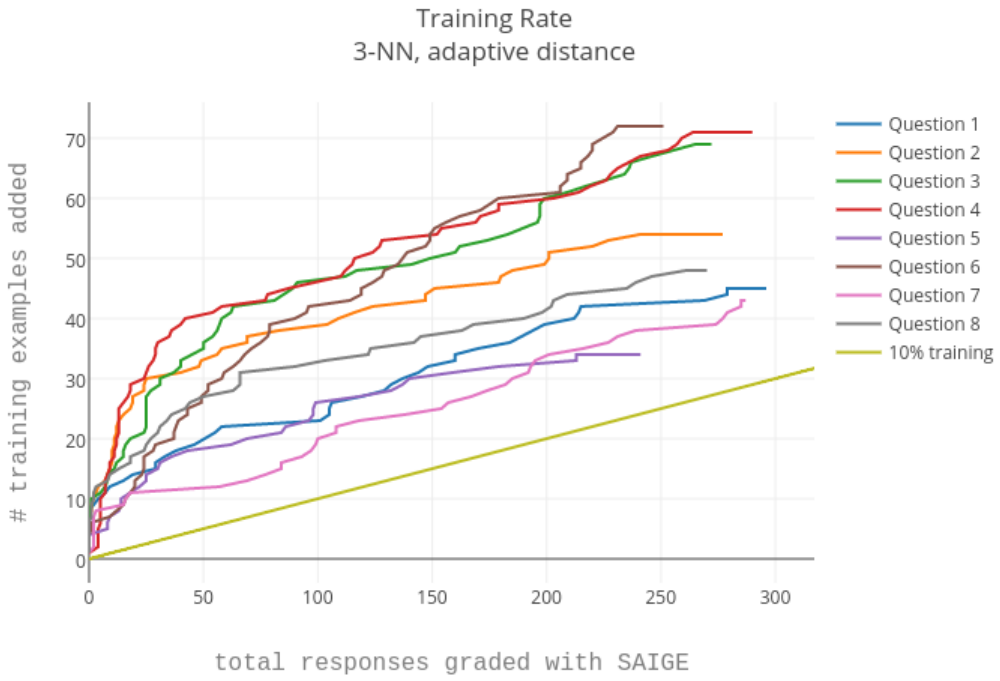


Figure 7.7: Training rate for KNN configuration with fewest errors

Figure 7.7 shows the training rate for all questions using k=3 AMD with unanimous vote. The x-axis shows the total responses graded for the question given the number of training examples. As the more responses are graded, the teacher should eventually reach a limit on the number of manually graded training examples. The y-axis represents the number of training examples the teacher has graded for a question. The total number of graded responses should increase between each training example, with the graph eventually asymptoting. These lines should fall under the 10% line at 10% training examples (between 26 and 36 examples) to meet our goal. No line falls under the 10% line at any point. Question 5 is the closest to achieving asymptotic training. Most of the questions have a decrease in slope over time, though the graphs never asymptote. This is not surprising, due to the constant need for training examples to resolve classifier confusion.

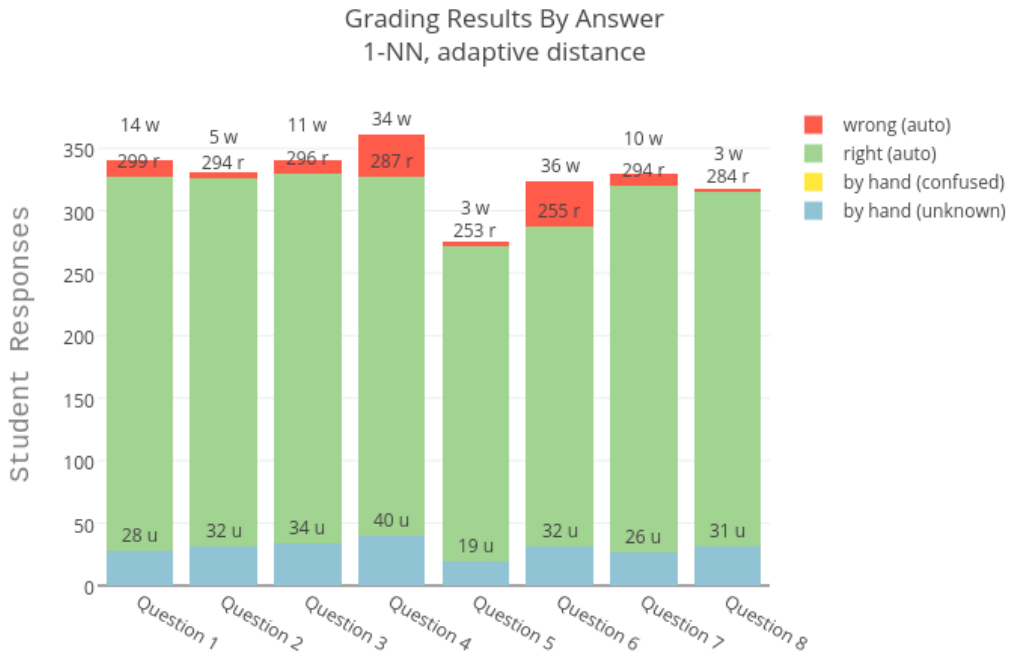


Figure 7.8: KNN configuration with fewest examples

Figure 7.8 is a graph of results using k=1 AMD. This combination achieved fewest training examples for all questions. Questions 2, 5, and 8 met the goal of <2% error and Questions 1, 3, and 7 had ≤4% error. Questions 4 and 6 had over 9% error, most of which was caused by ambiguity and stroke perversity. Seven of eight questions required <10% training examples, with Question 4 requiring slightly more (11%). With k=1, no voting is required to determine class, and no classifier confusion occurred to require additional examples.

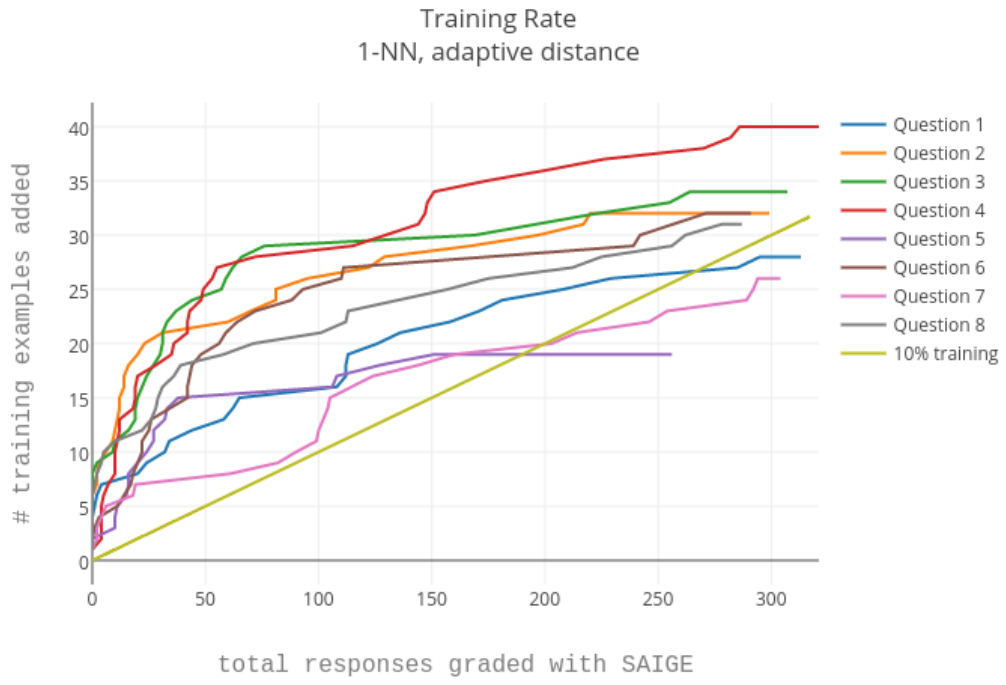


Figure 7.9: Training rate for KNN configuration with fewest examples

Figure 7.9 shows the training rate for all questions using k=1 AMD. Three questions fall under the line after approximately 10% training examples. Question 5 is the most asymptotic. Most of the graphs begin to approach asymptotic training, but stroke perversity due to meaningless strokes prevents true asymptotic training. Meaningless strokes were not learnable, and any response with meaningless strokes had to be manually graded and added as a training example.

Version Space

The version space algorithm builds a set of non-overlapping feature bounding boxes, or hypotheses, based on the observed training examples. These hypotheses are then used for classification by determining into which hypothesis a new example falls. SAIGE uses hypotheses as measures of certainty such that an example falling outside any hypothesis indicates a high

level of uncertainty. Traditionally the bounding boxes will be formed by bounding all the training examples as generally or specifically as possible without overlapping hypotheses. We adapted this basic algorithm to include an incremental training paradigm with two adaptations, swelling boundaries and splitting hypotheses. When SAIGE encounters a new example with no matching hypothesis it creates a new hypothesis for the example with slightly expanded bounds to catch ungraded data. We call this expansion "swell." When a hypothesis receives a new example, if needed it expands its boundaries to fit the new example exactly, and then swells its boundaries to increase generalization.

When hypotheses begin to overlap, SAIGE splits them and recalculates new, distinct bounds. Hypothesis overlap such that classes are nondisjoint causes classifier confusion, and splitting hypotheses limits classifier confusion and reduces errors by guaranteeing disjoint hypotheses. Figure 7.10 shows an example of how hypotheses are created and adapted.

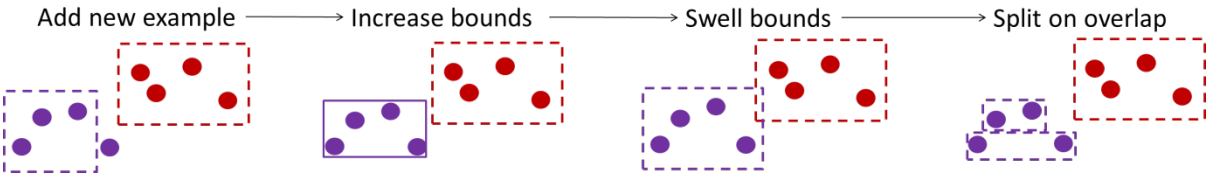


Figure 7.10: Adapting version space Hypotheses

We swell boundaries after a boundary change from splitting or adding a new example. We tested two different types of boundary swell: set swell and growing swell. Set swell simply increases the bounds by a predefined amount. Growing swell calculates the width of a feature and swells the boundaries by some fraction of the width. Larger swell values allow faster generalization but also increase the probability of hypothesis overlap.

Figure 7.11 shows the results of grading eight test questions with version space using no swell and no split. For this test we allowed hypotheses to overlap, and if an example fit two overlapping hypotheses SAIGE requested training on that example. Not splitting hypotheses showed us which questions were most likely to have nondisjoint hypotheses. This was a control group which gave a baseline for improvement. The number of training examples is high, between 23% and 82%, much greater than the10% goal. Questions, 1, 4, and 6 each have a large number of training examples due to classifier confusion from nondisjoint classes. The amount of error is variable, between 0.3% and 20%. Four questions had error under the 2% goal, and these questions also had the least amount of classifier confusion.

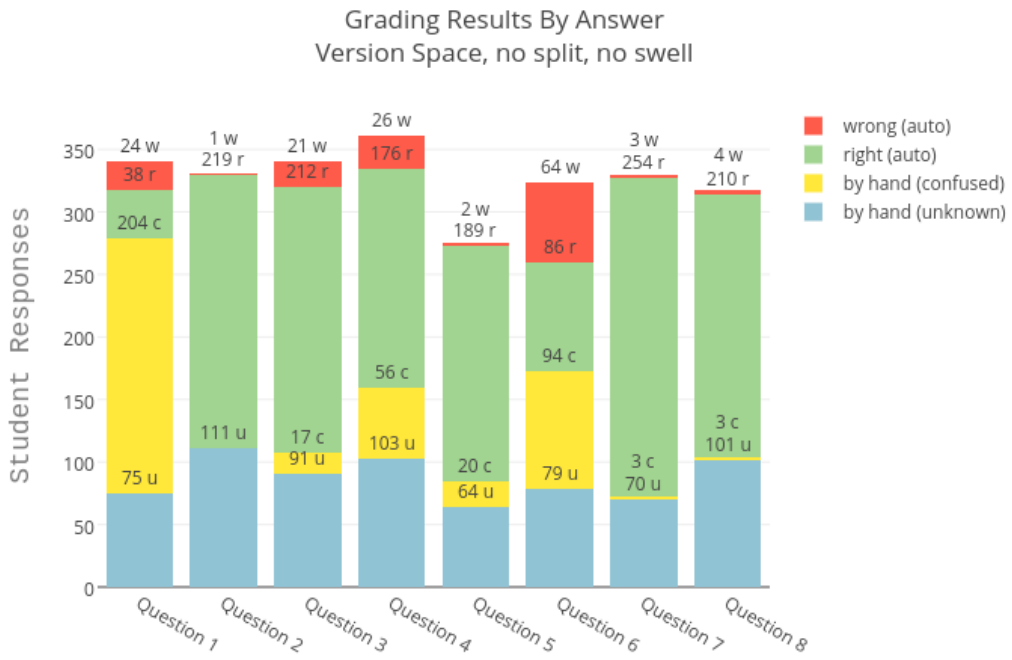


Figure 7.11: Baseline results for version space with eight questions

Figure 7.12 shows the total responses with errors, responses used as training examples, and responses autograded correctly for Question 1, given as percentages. This question had the lowest number of autograded responses and the highest number of classification confusions, and

had a moderate number of errors and examples compared to the other questions. We tested ten different swell combinations. The table contains the best results.

	Total Errors	Total Examples	Autograded correctly
Swell=None, No split	7.04%	81.82%	11.14%
Swell=None, Split	0.29%	45.75%	53.96%
Swell=0.02*w, Split	0.59%	34.60%	64.81%
Swell=0.05*w, Split	0.88%	27.27%	71.85%
Swell=3.0px, Split	1.17%	27.86%	70.97%

Figure 7.12: Version space adaptation results for Question 1 (341 responses)

Most of the training examples for the no-split configuration were a result of classifier confusion. The responses contained a large amount of stroke overlap and perversity which created ambiguity. Splitting hypotheses drastically reduced the number of required training examples and errors for this question. A small growing swell performed slightly better than a set 3px swell. We tried larger swell configurations, but the results were not as good. When swell grew larger, overlap occurred more often, creating more splitting and more required training. We considered using a maximum number of splits per class, but this did not guarantee that classes would remain disjoint, so more classifier confusion would have occurred.

The best version space combination used a growing swell of 0.05. The results are shown in Figure 7.13. This configuration allowed incremental expansion of bounding boxes to generalize slowly. Errors for each question were <1%, within our goal of 2%. Total training examples fell between 17% and 43% compared to the goal of 10%.

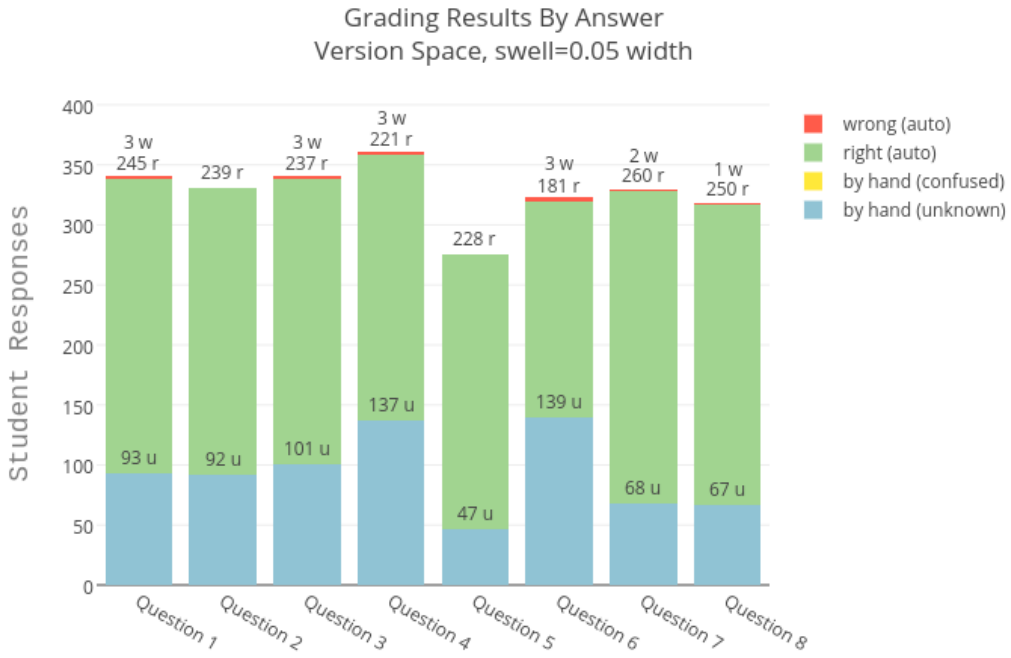


Figure 7.13: Version space configuration with best results

Additional training was required when there was human error while grading or stroke ambiguity, as explained previously with regards to KNN. When hypotheses overlapped, SAIGE would split a hypothesis into two smaller hypotheses which exactly fit their examples without generalization, which results in many small hypotheses instead of few large hypotheses. The smaller hypotheses were not general enough to recognize new strokes, and so additional training was constantly required. Version space is not robust with regards to noisy or ambiguous data, and our data is perverse in many ways.

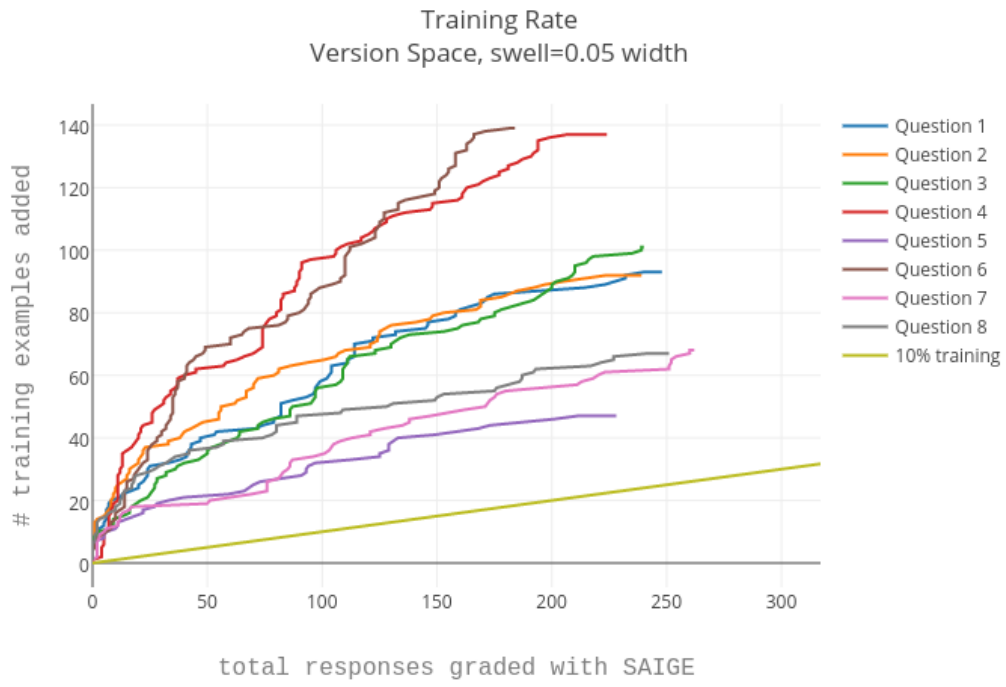


Figure 7.14: Training rate for version space classifier with best results

Figure 7.14 shows the training rate for all questions using a growing swell of 0.05. Question 5 is nearest to asymptoting of all the questions, but no line falls under the 10% line at any point. All the questions have a decrease in slope, but the decrease is gradual such that each maintains a mostly linear rate. This may be an artifact of constantly splitting hypotheses and thus creating a constant need for additional training examples. Stroke ambiguity plays a role in this, as they are the reason for overlap.

Classifier Comparison

The results for each classifier were mixed, with each configuration providing some benefits and some disadvantages. There was a tradeoff between number of examples and total errors in each case, and often the amount of error and training was heavily dependent on the question and responses.

KNN with k=1 AMD achieved the goal of <2% error for three of eight questions, and achieved <5% error for six of eight questions. This classifier met the goal of under 10% training for seven of eight questions, with the remaining question requiring 11% training. This classifier was the closest to achieving asymptotic training, had the lowest amount of training, but did not meet the 2% error goal for a majority of the questions.

KNN with k=3 AMD with unanimous vote achieved the goal of ≤2% error for seven of eight questions. The remaining question had 9% error. The classifier did not meet the goal of <10% training, with a range of 12-22% training. This classifier had low error overall, but did not achieve asymptotic training and did not meet the goal of 10% training.

Version space with growing swell=0.05 achieved the goal of <2% error for all the questions. This classifier did not meet the goal of <10% training, with a range of 17-43% training. This classifier had the lowest error overall, but the highest amount of training, and it was not able to achieve asymptotic training.

There was a constant tradeoff between the number of errors and the number of examples based on classifier and the stopping criteria chosen. Our attempts to improve stopping criteria also did not provide universal improvement, but differed from question to question. We discovered that how well a classifier performs depends on the question and answer as much as any tweak to SAIGE itself. Responses contained nondisjoint classes, context was sometimes more important than absolute location, and general perversity of strokes prevented SAIGE from obtaining asymptotic training. Incremental training was also a problem. Starting with only a few examples does not provide the necessary base for learning a problem well.

Neither classifier proved to have a significant advantage over the other. KNN provided SAIGE with a simple classifier for stroke comparison. It did surprisingly well for its simplicity,

but the relative positions of strokes proved challenging for generalization. Version space allowed SAIGE to use hypothesis bounds to identify stroke features. However, version space has an inherent inability to deal with noisy data, so classification confusion arose when there was ambiguity or human errors grading the question. It also required significantly more examples than KNN because after splitting hypotheses more training may be required to fit the data.

8. User Interface

SAIGE required an interface which would support semi-automated grading with incremental training while keeping burden of use low. Most manual grading is done by viewing handwritten responses one at a time. We used this premise when designing our interface, and decided on a "get-one-and-grade-it" approach. In this interface a single response is displayed in the grading view. Grading involves attaching feedback items to strokes in a 1:1 relationship. This interface was attractive because it is simple to understand and easy to build, and we could see and learn about individual responses.

Implementation

We give an overview of SAIGE in Chapter 2, in which we provide an example of use from question creation to the end, when a student views the graded response. In this section we explain the specifics of the grading interface by walking through the grading process step-by-step. Figure 8.1 shows the interface the teacher sees when they begin grading a response. Specific areas of interest are marked in red and labeled.

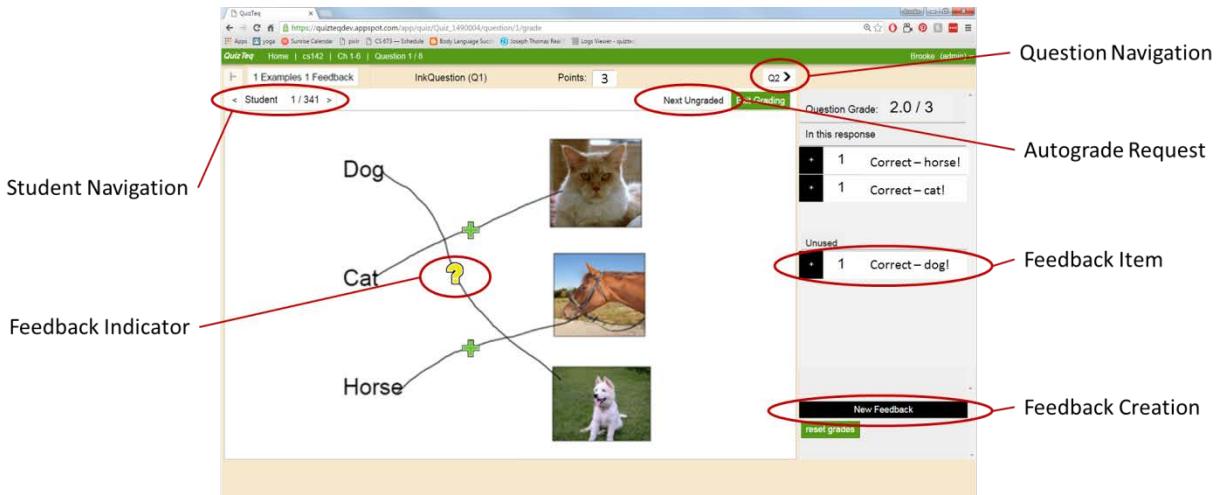


Figure 8.1: Grading Interface

The teacher is in the middle of grading a response. He has created feedback using the feedback creation button, and has graded two of the three strokes. Each stroke has a feedback indicator which marks the stroke as graded or ungraded. A yellow question mark indicates an ungraded stroke. A green plus symbol indicates a positive value for the stroke. Not shown in this response are a red minus, which indicates a negative value for the stroke, and a blue circle, which indicates a zero value for the stroke. Seeing the indicators, the teacher knows which strokes have been graded. When he wants to see which feedback item has been attached to which stroke, he clicks on a feedback indicator to select the stroke. If the stroke has been graded, the attached feedback item is highlighted on the right.

The teacher wants to grade the ungraded stroke, so he clicks on it to select it. He then views the unused feedback items on the right and sees the one he wants to attach. He does so by clicking on the black plus button on the feedback item. The feedback item moves from the unused section to the "in this response" section and the stroke's feedback indicator changes to a green plus symbol. When feedback is attached to a stroke, the stroke stores the id of the feedback item. Storing the id means future changes to a feedback item during grading will appear for previously graded strokes.

Now the response is completely graded, and the teacher is ready to grade the next response. He can choose to navigate through the responses manually using the student navigation, or he can use SAIGE to semi-automatically grade responses by clicking the autograde request button. The autograde request automatically grades as many responses as possible and then returns the next incompletely graded response. Choosing to autograde will send the response to SAIGE, where it processes the labeled strokes and adds them as training examples to its classifier. SAIGE keeps track of which response was graded last, so after training

has been updated SAIGE reviews the next response to see if it has been graded. SAIGE looks at each stroke in the response, and if any is not labeled with a feedback item, SAIGE tries to assign a label using the classifier. If the stroke remains unlabeled, SAIGE will send the response to the teacher. Otherwise, it reviews the next response in the list, and so on. After reaching the end of the list, SAIGE goes through every response once more to verify all responses are completely graded before finishing. This can take several seconds, so a message for the teacher appears to indicate when SAIGE is autograding, and disappears when the next response is available. When the teacher is finished grading the responses for a question, he can use the question navigation buttons to move to the next question.

Using SAIGE - Analysis of Paradigm

One of our metrics for success is that SAIGE provides a significant time reduction compared to manual grading, specifically limiting time to an average 30 seconds per question including wait time. We ran an informal time test by semi-automatically grading a question, recording the grading time for each response. Grading time included time waiting for SAIGE to autograde and return the next response to manually grade. We used SAIGE to grade a set of 341 responses to a question from start to finish, including creating feedback, and recorded the time spent interacting with each response. We then regraded every response manually, recording the total grading time for the question, which illustrated the true time gain SAIGE provides.

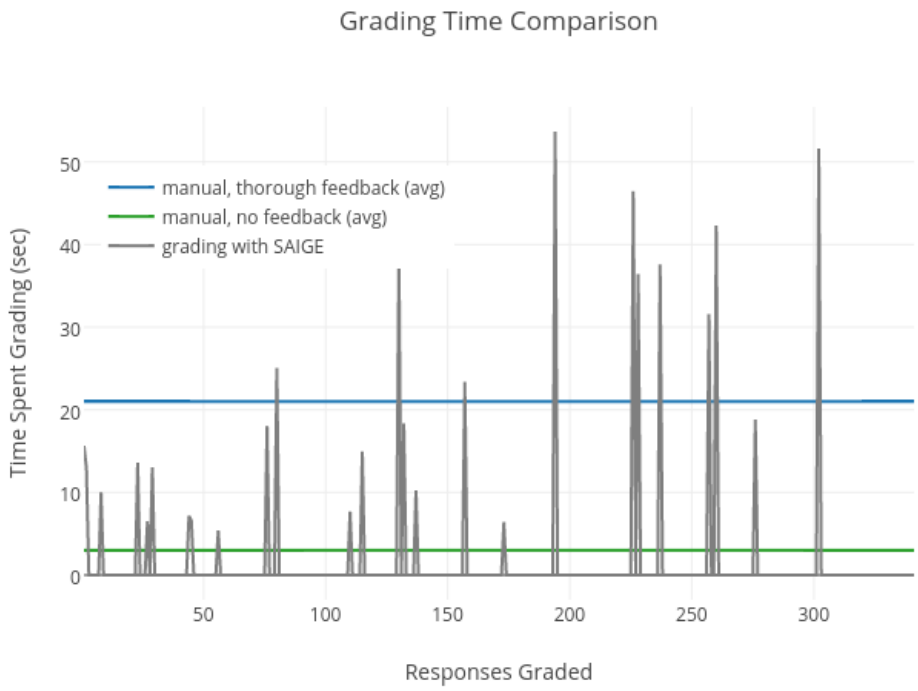


Figure 8.2: Time spent (in seconds) grading responses manually and using SAIGE

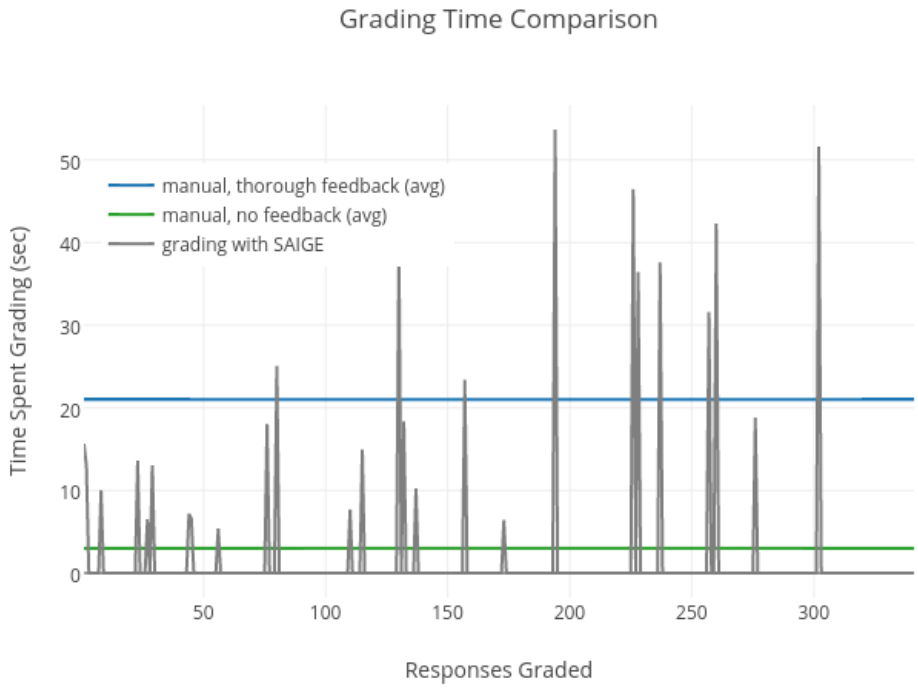


Figure 8.2: Time spent (in seconds) grading responses manually and using SAIGE

shows the time spent grading each question manually and using SAIGE. We recorded times for both careful and quick manual grading for comparison. We did not record individual times for each response during manual grading, thus the graph shows a line to represent the average time spent per response for manual grading. We individually recorded time spent interacting with each response with SAIGE, illustrated by the gray line in the graph. On this graph, any response which required no interaction with the user was recorded at zero. Interaction time for a manually graded response combines the time for two processes, actual grading time and wait time. Grading time spanned from the time a response loaded to the time a teacher requested the next response. Wait time was the time spanned from the time a teacher requested a response to the time the question loaded. We chose not to separate grading time and wait time because our measuring mechanism did not support it easily, and because using these times together gave a more accurate representation of true time spent in this grading experience.

Carefully grading every response manually took two hours for a single question, an average of 21seconds per question, with some speedup over time as the grader grew more familiar with the responses. This manual grading experience included carefully reviewing each response and giving feedback for each stroke in a response, and represents a possible scenario for a teacher who wants to give thorough feedback. We simulated quickly grading a question by reviewing each response and writing down a score for the question, giving no feedback. This fast approach took an average of 3 seconds per question including an initial learning curve for a total of 17 minutes, and provided no feedback. By contrast, using a detailed grading approach to grade the same question with SAIGE took a total of ten minutes, and required grading only 27 responses, approximately 8% of the total responses for the question. Both manual grading approaches required more time than SAIGE, because although one manual approach was

significantly faster than the other, it remained a linear grading approach which required viewing every student.

The responses with the longest grading time were responses which were ambiguous, had perverse strokes, or required creating new feedback. Overall, the average time for each question was 21.88 seconds, and the median time was 17.8 seconds. The time to grade falls well within the goal of 30 seconds per question even before we addressed any stroke perversity to improve grading time. Responses with a higher number of perverse strokes required more grading time at this early stage of development. This time was reduced by including the ability to ignore meaningless strokes as explained in the previous chapter, and interaction time could be further reduced by optimizing SAIGE to minimize wait time.

Grading with SAIGE meant spending time giving in-depth feedback for a few responses, for an overall decrease in grading time. Of note was that while semi-automatically grading with SAIGE, we were still able to see individual student responses, which provided insight and occasionally humor while grading. SAIGE's interface was reasonably simple to use and allowed teachers to gain insight from responses and provide specific feedback. We were also able to discover the weaknesses of this paradigm.

One weakness of SAIGE's interface is that when grading a partially-autograded response it was natural to assume the autograding was correct, so autograding mistakes were easily missed and never corrected. For instance, in the response shown in Figure 8.3 strokes A, B, and C should each receive a unique feedback item with a positive value. At first glance it seems the autograder correctly identified the strokes and assigned them correct feedback items. However, strokes A, B, and C received the same feedback during autograding: "int needed before force." The user was restricted to assigning one feedback to one stroke, but SAIGE's method of

classification meant it was not under the same restriction, so this error could easily occur. We did not notice the error until we manually regraded all the responses and verified each stroke individually.

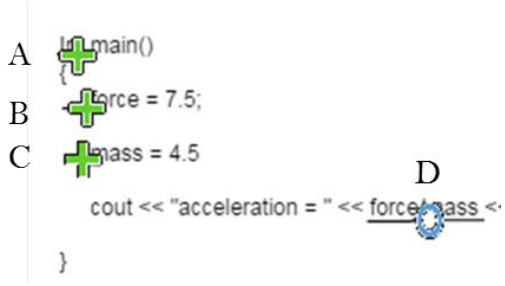


Figure 8.3: Non-obvious grading errors

Another problem with one-at-a-time grading is that of grading inconsistency, which is a problem in manual grading that this interface does not fully overcome. Inconsistency when grading confuses SAIGE, and occurs because of opinion drift or human error. This is particularly a problem with ambiguous responses, which we frequently found in our data. We use the example in Figure 8.4 to illustrate each of these concepts.

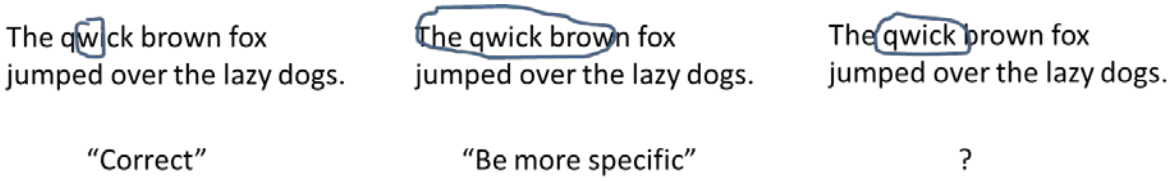


Figure 8.4: Response ambiguity

This example asks that the students mark where errors occur in the sentence. While grading, the teacher sees the illustrated responses: one very precise, one very imprecise, and one in between. The teacher grades the first and second responses easily: one correct, one too imprecise. The third response is ambiguous. After some thought the teacher decides that this response is specific enough to be marked correct. The teacher continues grading and eventually

sees another instance of the third response. Either because they forgot their prior decision or because they changed their mind, this time the teacher marks it as too imprecise. This occurred frequently as we used SAIGE. Viewing one response at a time made it easy to forget how an ambiguous response was previously graded. We frequently also experienced a drift in our perception of a response as we graded other responses. We also found our attention wander while grading, which caused us to make mistakes due to inattention. These situations caused grading inconsistency which confused SAIGE.

Complications also arise when using a 1:1 rule for attaching feedback to strokes. Figure 8.5 shows three responses and the feedback used in each response. The feedback shown gives some insight into the undesirable results from using 1:1 feedback attachment.

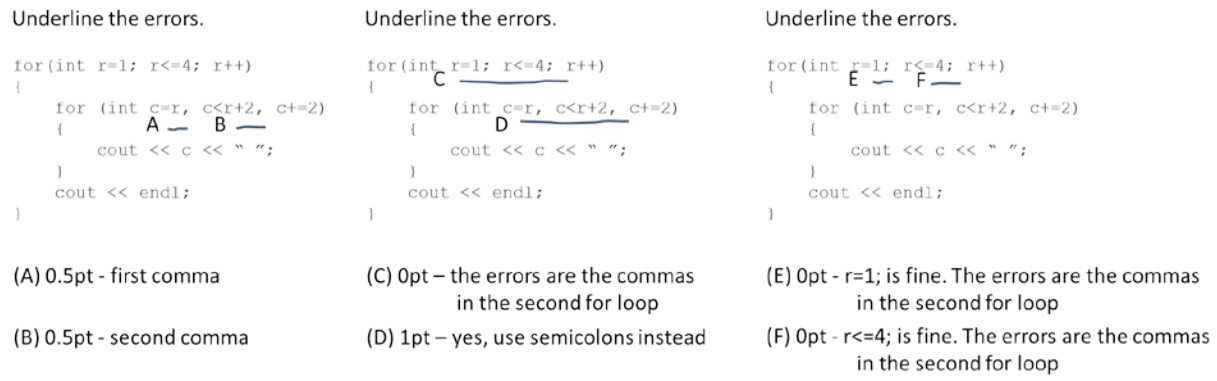


Figure 8.5: Feedback for three responses

In this problem, the teacher is looking for general comprehension about code syntax. Specifically, she wants to see that her students know to use semicolons instead of commas within a for loop declaration, and asks them to underline the errors in a piece of code. The teacher sees the first response and sees the student has marked two locations with strokes A and B. Because she cannot reuse feedback, it is necessary for her to divide her feedback and the points for each correct stroke. Since the student understands the problem, she does not create detailed feedback

as she grades these strokes. The resulting feedback text explains which stroke the feedback corresponds to, without deeper explanation. Next, the teacher sees the second response, with strokes C and D. Stroke C is incorrect, and the teacher explains why as she assigns feedback. Stroke D encompasses both of the errors the student should have marked, and the teacher decides to give the student points for the stroke. She would like to recycle the feedback items she used for strokes A and B, and edit the feedback text for better explanation, but it is not possible to attach multiple feedback items to a stroke. Instead she creates a new feedback item with a single explanation for the single stroke. Now the teacher begins grading the third response. She sees that both strokes E and F are incorrect, and would like to recycle her feedback from grading stroke C for both of them, since strokes E and F deserve the same point value and explanation as stroke C. However, it is not possible for the teacher to attach one feedback item to multiple strokes in the same response. She now must create two unique but redundant feedback items to grade the strokes. In the end the teacher has created twice as much feedback as she would truly need, and the feedback is redundant.

Evaluation of Success

The SAIGE interface was simple to build and reduced manual grading for a large class from two hours to ten minutes. Grading each question took an average of 21.88 seconds, within our goal of 30 seconds. The one-at-a-time interface allowed the grader to see and understand individual responses, and feedback was specific to the real responses given. However, there were significant disadvantages to this interface. It was difficult to quickly and easily notice autograding errors. One-at-a-time grading means that human error and opinion drift commonly caused grading inconsistency.

9. Summary and Evaluation of Success

Teachers prefer open-ended quizzes because they provide a richer experience than responses with predetermined answers. However, open-ended quizzes are laborious to grade, especially as class size increases. Existing methods to automate grading reduce teacher workload with limited success, but no method directly addresses questions with freeform drawing responses. We used stroke recognition and interactive machine learning concepts to build SAIGE. SAIGE is a novel quizzing format which uses digital ink-based questions and semi-automated grading to reduce grading overhead for open-ended quizzes. We set out to discover if SAIGE could overcome the grading challenges that teachers face, namely grading speed, grading consistency, and providing high-quality questions and effective feedback.

SAIGE was required to meet four goals to be considered successful. The number of training examples must not exceed 10% of all responses for datasets of more than 50 responses, and the training must asymptote. Number of training examples must be <10% to show that the teacher's workload is significantly reduced, and training must asymptote to ensure the grading becomes a sublinear problem. The number of responses with autograding errors must not exceed 2% of all responses. Human graders are rarely if ever perfectly accurate, so we allow a small margin for classification errors. Actual grading time must not exceed an average of 30 seconds per question, to guarantee a reasonable amount of time spent grading. We evaluate the number of training examples, asymptotic training, and classification errors based on data gained through the classifier selection tests described in Chapter 7. We evaluate actual grading time using data gained from the interface usage tests described in Chapter 8.

Choosing the Best Features

We investigated several feature combinations to find an ideal set of features for accurate stroke classification with few examples. These feature sets included using n-points along the stroke, endpoints, bounding box, and hit points of a stroke. We discovered there is not one best feature set to minimize examples and maintain high accuracy, but that we can differentiate strokes into lines, circles, and paths. We implemented automatic stroke differentiation, which allows some reduction of training examples by emphasizing the most important features for a given stroke.

Perversity of Strokes

We discovered stroke perversity in the form of stroke direction, split strokes including multistrokes and crossout strokes, and meaningless strokes. We addressed each of the perversities. We normalized stroke direction to reduce the number of examples and errors in training and classification. We created algorithms to automatically identify and merge multistrokes and crossout strokes. Merging multistrokes and normalizing stroke direction eliminated errors and reduced training examples from 20 to 16 for our test data. Due to the complexity of crossout strokes, we removed them from our data but recognized that they were present in 5% of responses, which we consider prevalent enough to merit future exploration. We allowed teachers to mark all meaningless strokes at once with an "ignore" flag, which improved time to grade each question. Meaningless strokes also require further attention to sufficiently reduce teacher workload.

Selecting a Classifier

We investigated two classifiers which seemed natural candidates for this task, KNN and version space. We implemented multiple versions of each, using generalization with stopping criteria to reduce the number of training examples. We found that neither classifier was superior to the

other, and found a tradeoff between reducing errors and reducing training examples. This was a clear illustration of the "no free lunch" theorem in this field of study.

SAIGE with KNN (k=1 AMD) required fewer than 10% training for seven of eight questions, and required 11% training for the remaining question. SAIGE with version space (growing swell=0.05) required between 17% and 43% training examples. The range of training examples required is indicative of how much the amount of training depends on the data given. Questions with simple responses and little ambiguity required few examples. Complex responses and responses with ambiguity required more training examples due to classifier confusion.

No classifier implementation was able to achieve asymptotic training for any question. Each implementation did show a decrease in slope as training examples were added, but after the decrease the slope remained linear. This result was in part due to classifier confusion that required continual training to resolve ambiguity. The results also resulted from stroke perversity which we did not fully resolve. Responses with meaningless strokes remained unlearnable, and required a teacher to view the response and verify the meaningless strokes.

SAIGE with version space (growing swell=0.05) had <2% error for all questions. SAIGE with KNN (k=3 AMD unanimous) had ≤2% error for seven of eight questions, with 9% error on the remaining question. This outlier question illustrates how much error can depend on algorithmic ambiguity of the training data and consistency of the grader. Figure 9.1 shows this question and two graded responses which should have received the same feedback. The response on the left was graded correctly, but the response on the right was graded incorrectly. Multiple grading errors of this kind caused this response to become algorithmically ambiguous with KNN.

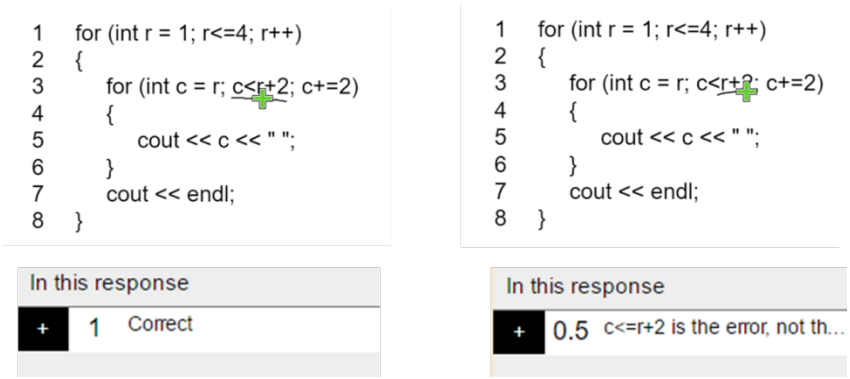


Figure 9.1: Human error can create algorithmic ambiguity

Both classifiers struggled when faced with ambiguity in the training data. KNN was able to achieve the goal of 10% training for most questions, at the cost of having more errors. Version space was able to achieve the goal of 2% errors, at the cost of requiring more training examples.

User Interface

We built the digital ink quiz platform based on a one-at-a-time grading paradigm. The interface was simple to build and meant to be similar to a manual grading experience. Coupled with the semi-automated grading, grading with this interface was fast. Grading with SAIGE took an average of 21.88 seconds, better than the goal of 30 seconds. The total time to grade 361 responses to a single question with SAIGE was ten minutes. For comparison, manually grading every response took two hours. The interface did limit the number of training examples we could provide the classifier for learning, and did not fully overcome the challenges a manual grading experience includes.

Overall Evaluation of Success

The individual strokes within student responses proved to be one of the most influential factors in how much training data was required. Some questions had responses which had high levels of

success, and some proved more challenging to grade. SAIGE could not achieve asymptotic training or fully achieve both 10% training and 2% errors.

SAIGE was unable to automatically grade some strokes due to unresolved stroke perversities. Meaningless strokes are an unresolved perversity which contributed to SAIGE's inability to achieve asymptotic training. Crossout strokes are an unresolved perversity which were present in 5% of student responses and must be addressed.

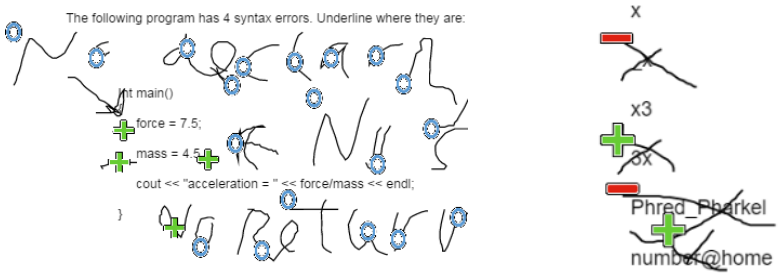


Figure 9.2: Unresolved perversities: meaningless strokes (left) and crossout strokes (right)

A tradeoff between training examples and classification accuracy could not be overcome, especially with our grading paradigm. There is a natural need for a certain amount of training data in any machine learning problem in order to achieve a given accuracy. More training data usually leads to higher accuracy. Our UI allows us to add only one training example at a time, and in order to reduce manual grading overhead we created a goal to minimize the number of training examples SAIGE receives. However, we were not able to sidestep the effects of the inherent accuracy tradeoff, thus when we reduced training examples classification errors increased, and vice versa. Our one-at-a-time grading paradigm was unable to meet the performance goals we set.

The one-at-a-time grading paradigm also resulted in grading inconsistencies common to manual grading. These problems included human error and opinion drift, in which a teacher

changes his mind over time. Grading inconsistency most often occurs when ambiguity is present such as in Figure 9.3, where the third response may be correct or it may be too unspecific.

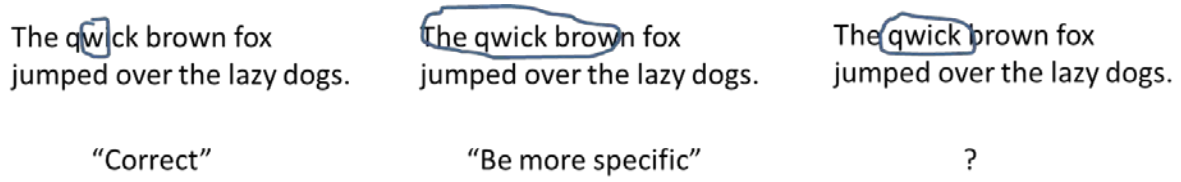


Figure 9.3: Ambiguity which leads to grading inconsistency

Conclusion

We laid the groundwork for additional exploration in the grading area. SAIGE was effective at semi-automatically grading a large number of student responses, with up to 10x speedup in grading time, low errors, and small amounts of training data. Although it did not attain full success, SAIGE was able to correctly autograde the majority of responses. It also met the goal of taking <30 seconds to grade each question, and needed only ten minutes to grade an open-ended question which took two hours to grade manually. Additional work is necessary for SAIGE to be an effective quizzing tool. The one-at-a-time grading paradigm was not fully sufficient for a good grading experience, and so an alternate grading paradigm is necessary. However, SAIGE proved it is possible to automatically grade digital ink quizzes with some success and create a semi-automatic grading experience for teachers.

10. References

1. Anthony, L., & Wobbrock, J. O. (2010). A lightweight multistroke recognizer for user interface prototypes. In *Proceedings of Graphics Interface* (pp. 245-252).
2. Bellman, R. E. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.
3. Brooks, M., Basu, S., Jacobs, C. & Vanderwende, L. (2014). Divide and correct: Using clusters to grade short answers at scale. In *Proceedings of the 1st Conference on Learning at Scale* (pp. 89–98).
4. Ciresan, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2010). Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22 (12), 3207-3220.
5. Conole, G. & Warburton, B. (2005). A review of computer-assisted assessment. *Research in learning technology*, 13 (1), 17–31.
6. Fails, J., & Olsen, D. R. (2003). A design tool for camera-based interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 449-456).
7. Fails, J. A., & Olsen, D. R. (2003). Interactive machine learning. In *Proceedings of the 8th International Conference on Intelligent user interfaces* (pp. 39-45).
8. Fuccella, V. & Costagliola, G. (2015). Unistroke Gesture Recognition Through Polyline Approximation and Alignment. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 3351-3354).

9. Gibbs, G., Simpson, C., Gravestock, P. & Hills, M. (2005). Conditions under which assessment supports students' learning. *Learning and Teaching in Higher Education*, 1 (1), 3-31.
10. Hady, M. F. A., & Schwenker, F. (2013). Semi-supervised learning. In *Handbook on Neural Information Processing* (pp. 215-239).
11. Jordan, S. & Mitchell, T. (2009). e-Assessment for learning? The potential of short-answer free-text questions with tailored feedback. *British Journal of Educational Technology*, 40 (2), 371–385.
12. Kara, L.B. & Stahovich, T.F. (2004). An image-based trainable symbol recognizer for sketch-based interfaces. In *AAAI Fall Symposium* (pp. 99-105).
13. Liu, Z., Zhao, X., Zou, J., & Xu, H. (2013). A semi-supervised approach based on k-nearest neighbor. *Journal of Software*, 8 (4), 768-775.
14. McMillan, J. (2001). Secondary teachers' classroom assessment and grading practices. *Educational Measurement: Issues and Practice*, 20 (1), 20-32.
15. Patel K., Fogarty J., Landay J. A. & Harrison, B. (2008). Investigating statistical machine learning as a tool for software development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 667-676).
16. Ritter A. & Basu S. (2009). Learning to generalize for complex selection tasks. In *Proceedings of the 14th international conference on Intelligent user interfaces* (pp. 167-176).
17. Rubine, D. (1991). Specifying gestures by example. *Computer Graphics* 25 (4), 329-337.

18. Taele, P., & Hammond, T. (2015). BopoNoto: An Intelligent Sketch Education Application for Learning Zhuyin Phonetic Script. In *Proceedings of the 21st International Conference on Distributed Multimedia Systems* (pp. 101-107).
19. Tanha, J., Someren, M. V., & Afsarmanesh, H. (2011). Semi-supervised self-training with decision trees: An empirical study. In *3rd IEEE International Conference on Intelligent Computing and Intelligent System (ICIS)* (pp. 669-704).
20. Tanha, J., van Someren, M., & Afsarmanesh, H. (2015). Semi-supervised self-training for decision tree classifiers. *International Journal of Machine Learning and Cybernetics*. doi:10.1007/s13042-015-0328-7
21. Tyler, C. (2013). Professor spearheads new system to streamline grading, save time. *The Daily Californian*. Retrieved September 14, 2015, from http://www.dailycal.org.
22. Wang, Y., Xu, X., Zhao, H., & Hua, Z. (2010). Semi-supervised learning based on nearest neighbor rule and cut edges. *Knowledge-Based Systems*, 23 (6), 547-554.
23. Wobbrock, J.O., Wilson, A.D. & Li, Y. (2007). Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology* (pp. 159-168).
24. Zhu, J., Wang, H., Tsou, B., & Ma, M. (2010). Active learning with sampling by uncertainty and density for data annotations, *IEEE Transactions on Audio, Speech, and Language Processing*, 18 (6), 1323-1331.