Investigating Intuitions and Predicting Success using Fine Grained Student Code Snapshot Data

Henry Maier

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-128
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-128.html

May 14, 2022

Investigating Intuitions and Predicting Success using Fine Grained Student Code Snapshot Data

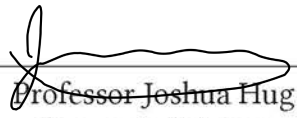
by Henry Maier

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II.**

Approval for the Report and Comprehensive Examination:

Committee:



Professor Joshua Hug
Research Advisor

12 May 2022
Date

* * * * * * *

Paul Hilfinger

Professor Paul N. Hilfinger
Second Reader

12 May 2022
Date

Investigating Intuitions and Predicting Success using Fine Grained Student Code Snapshot Data

Henry Maier
University of California, Berkeley
henrymaier@berkeley.edu

ABSTRACT

At UC Berkeley, unprecedented growth and demand for Computer Science education has resulted in difficulties assessing how students navigate introductory Computer Science courses. Instructors are left using intuition, low quality survey data and information relayed by large course staffs to determine pain points and identify students at risk of underperforming. In response, previous work developed Snaps [10], a tool for CS61B (CS 2) at UC Berkeley that captures fine-grained snapshots of student work as they complete assignments. We build simple metrics for assignment completion time and working habits using the Snaps dataset from CS61B Spring 2021. Using these metrics, we investigate individual associations between these metrics and performance, as measured by scores on exams and final course grade. We find rough associations that confirm intuitions relating working time and working habits to performance. Specifically, we find that students who spend longer and start later on assignments tend to perform worse in the course. Additionally, we develop and evaluate models that use our metrics, as well as grade data from CS61A (CS 1) at UC Berkeley as features and find an early prediction model based on data from the first 2 weeks of the course for exam points ($R^2 = 0.656$) and final course grade ($R^2 = 0.585$) with limited accuracy.

1 INTRODUCTION

In the past decade, Higher Education Computer Science programs have seen rapid enrollment growth and never before seen high demand for quality education [8]. To keep pace with this growth and demand, programs have employed a variety of strategies to continue to deliver high quality education. At UC Berkeley, instructors have turned to automated grading and delivering course offerings online to handle large class sizes. Unfortunately, these strategies result in limited one on one attention for students.

At any given moment, instructors have only a few data points that provide insight into how students are doing in the course: exam results and performance on assignments. In order to get insight into challenging concepts and student work habits, some instructors employ techniques such as weekly student surveys, and asking for feedback from course staff members about their interactions with students. These techniques give a biased view of how students are doing in the course: surveys suffer from response biases and non-response bias, and course staff feedback focuses on the proportion of the class that is consistently engaging with course resources, in addition to being clouded by the biases of the course staff members.

While at some institutions, the failures of the above mechanisms are not severe due to an abundance of staff capable of providing more personalized attention and time, Computer Science programs operating at scale are without the time to provide a level of individualized attention that would result in high quality feedback to instructors. At UC Berkeley, there are as many as 2000 and 1500 students enrolled in CS 1 and CS 2 respectively [1], with each class employing upwards of 60 course staff members.

With imperfect information and operating at scale, instructors of large classes are unable to make pedagogy decisions driven by data and must rely on rough metrics, intuition and experience alone. This is especially true when it comes to identifying struggling students in introductory courses. While current techniques (surveys, course staff feedback) can identify students in danger of

underperforming, many students appear solid according to these techniques only to do poorly on exams or large scale programming assignments.

Furthermore, without a strong signal, instructors are left with little feedback on how the decisions they make impact the performance of students until midterm exam or final grades are released, and even then, cannot reliably relate certain decisions to grade outcomes. Take, for example, an instructor who implemented a mechanism to encourage students to start assignments early (such as a "checkpoint", where a small portion of the assignment is due a week or so before the full assignment is due). There is simply not enough information for the instructor to reliably assess whether or not this mechanism is working to improve student performance or working habits.

In the 2020-2021 academic year, to address these issues, Snaps [10] was developed and deployed in UC Berkeley's CS 2 course (CS61B: Data Structures and Algorithms). Snaps is a piece of software that collects fine grained snapshots of student code as they work on programming assignments and makes these snapshots available to instructors and members of course staff. Each Snaps data point is a record of a student's entire code base at a given moment in time, and these records are taken roughly every few seconds while students work on assignments.

In this report, we utilize the Snaps dataset from the Spring 2021 semester to develop custom metrics to combat the lack of informative data available in introductory computer science courses, while keeping in mind the time constraints on instructors and course staff members. We then use these metrics to understand the interaction between student assignment completion times, work habits and performance as measured by points earned. Finally, we discuss the impact of our findings on current decision making by instructors at UC Berkeley and directions for future work.

2 RELATED WORK

2.1 Predicting Success

With the ability to accurately predict student success, instructors of large scale Computer Science courses would be able to focus course resources where they are most needed, performing timely interventions and updating course policies/curricula to provide more support for areas associated with poor performance. To illustrate, imagine a tool was built to detect when students were heavily relying on print statement debugging rather than using a debugger, and one found that not using a debugger was predictive of poor performance. An instructor could deploy content to encourage students to use a debugger, using the data from the tool to understand the effectiveness of this content or customize it for particularly debugger-averse students.

"Predicting success" is a broad term, and has been formalized in a number of different ways in other work. One approach has been to take data from performance on assignments and exams in prior courses, and try to use this as an indicator for performance on exams and assignments [5]. In fact, a large-scale analysis using UC Berkeley data concluded that a very strong predictor of a grade in an introductory computer science course is the grade received in the previously taken course [6]. Other approaches use non-grade factors from before beginning a course as predictors of success as measured by exam scores and assignment grades, such as mathematics background, prior programming experience, or even attitude towards learning Computer Science and success in the course [12]. While beneficial for generating lists of potentially struggling students very early on in a course, these approaches fail to incorporate data from the current course or give an instructor actionable insights for how to change their course.

There are approaches to predicting student success that have incorporated current data, using code snapshots and machine learning techniques to get accurate and early predictions for struggling students [4]. While these approaches do allow for timely intervention based on current course data, they rely on students actively creating code snapshots, which reduces accuracy on low-participation students. Additionally, the use of deep learning techniques reduces interpretability, essentially making the model a black box and resulting in difficulties providing actionable insights to instructors.

2.2 Automatic Snapshot Data

Other tools have been developed to automatically collect student code snapshots, and make the data available to instructors. As the tools have been developed at different institutions, they rely on certain portions of the course infrastructure, for example the Integrated Development Environment (IDE) that the course expects students to use while completing assignments, or the

version control system that is used for students to make code checkpoints. While there are differences in implementation details, these tools essentially provide the same data: timestamped snapshots of student code collected at semi-regular intervals that give a fine-grained view into how students complete assignments.

Yan et. al. considered a number of use cases for this data. First, in TMOSS, the snapshots were used for plagiarism detection between intermediary snapshots, rather than just final submissions as is standard practice [13]. Second, in Pensieve, the snapshots were used as a part of a tool developed to optimize pedagogical best practice in one-on-one feedback sessions between course staff and students [14].

Rodriguez-Rivera et. al. uses the source control snapshots to understand student progress in large scale classes. A tool is designed to aggregate the snapshot data to provide real time insights [9]. The tool surfaces metrics such as estimated time spent, progress based on the test suite, lines added/deleted and code similarity to instructors.

These previous approaches provide good motivation for the use of such a tool at UC Berkeley, and provide insight into potential pitfalls when analyzing the Snaps dataset. The details of the courses these approaches were used in must also be kept in mind as they differ in the scale and implementation details of the Snaps tool.

3 BACKGROUND INFORMATION

3.1 CS61B

CS61B: Data Structures and Algorithms is UC Berkeley's CS 2 course. Programming assignments are done in Java, and are mainly completed using the IntelliJ IDE. It is the second semester-long course in a series of three courses required in order to declare the CS major. A large portion of the students in a given semester have just completed the CS 1 course (which is taught in Python) and intend to declare the CS major, though there are students intending to major in other subjects (most notably Data Science, Cognitive Science, or some flavor of Engineering), or who have not taken UC Berkeley's CS 1 course.

A student's grade in CS61B during the Spring 2021 semester was determined by their performance on weekly Lab programming assignments, three written Homework assignments, two Midterm exams, a Final exam, and four large programming Projects [3]. Students additionally receive points for completing non-technical surveys throughout the semester about their experience with the course. Students' CS61B grades are primarily determined by their performance on exams, as median grades for other assignments are near 100%. That said, the programming projects represent the majority of the hours students spend on CS61B in a given semester. In Spring 2021, the projects students completed were:

- Project 0: 2048, in which students fill in methods in order to implement the game logic for a Java version of the popular 2048 mobile game.
- Project 1: Data Structures, in which students implement both an Array and Linked List backed version of a Deque interface. Students are also required to write tests to ensure correctness for their implementations.
- Project 2: Gitlet, in which students implement a simpler version of the Git version control system given descriptions of the functionality of commands, and vague guidance on the classes that should be written to implement these commands.
- Project 3: Build Your Own World (BYOW), in which students work with a partner to implement the generation of random 2D graphical worlds that subscribe to certain specifications, as well as the ability of a user to explore these worlds through keypresses. A graphical rendering engine is given in the starter code.

Students learn course material through lectures and optional weekly lab and discussion sections. Students can receive support on assignments by asking questions on a course forum, as well as attending Office Hours available many times throughout the week.

3.2 Snaps

The tool used to actually capture student data was the CS61B Snaps IntelliJ IDE plugin developed by Itai Smith and described in [10]. In the first lab assignment, students download necessary software to complete CS61B assignments, including installing and setting up the CS61B Snaps plugin. Throughout the course of the semester, students work on assignments in a local Git repository, which is connected to a remote Github repository that can be used to submit assignments to autograders. The Snaps plugin creates a separate local

Git repository that tracks and checks in intermediate changes, or "snapshots", of their code as they work through assignments.

The Snaps IntelliJ plugin checks in snapshots whenever IntelliJ "saves" code, which can happen due to changes made to a file open in IntelliJ, or when code is compiled [7]. As these snapshots are stored as Git commits, the plugin additionally stores the time at which the snapshot was made, and a list of the file names that have changed in that snapshot. In practice, the technique of using IntelliJ “save” events to trigger Git snapshots resulted in both very fine granularity of snapshots (some snapshots are taken within tens of seconds of each other) and low memory and compute constraints, as Git handles all of the change tracking and recording [2].

In the Spring 2021 semester of CS61B, we asked students to push their Snaps repositories consisting of the snapshots collected to a Github repository that instructors could access at four times throughout the semester (aligning with the deadlines of the four Projects).

The Snaps plugin was designed with simplicity in mind so that students could focus on actually completing the course, rather than setting up an experimental tool. In practice, some students had issues with setup or the plugin stopped working (for example, if a student switched computers at some point during the semester). We detail how we handle this inconsistent data in the next section.

3.3 Data Cleaning

Our dataset consists of 1550 Snaps repositories collected using the Snaps IntelliJ plugin during the Spring 2021 semester offering of CS61B. In practice, the Snaps plugin ended up being less reliable than expected, with some students running into errors due to setup problems or computer switching, and others neglecting to push their repositories following project assignments. There were also external factors that contributed to incomplete data, such as students dropping the course, taking the course for a Pass/Fail grade, or not completing assignments (per course policy, students were able to skip some Lab assignments without penalty).

In our analysis, we did not consider students who dropped the course or took the course for a Pass/Fail grade. While it is still important for instructors to get an understanding of the path these students took through the class, we measure success through final class performance and exam performance, so including these data only results in outliers and confounds relationships.

The nature of certain assignments additionally cause incompleteness and missing data. For example, Project 3 is open ended and gives students the freedom to create their own file structure. As we use file names to classify snapshots as belonging to a certain assignment, it is challenging to confidently classify a snapshot as being taken while a student works on Project 3. Additionally, Lab 4 and 5 both do not require students to edit code in IntelliJ, though some choose to. Finally, Lab 12 and Lab 13 are often skipped by students due to the course policy, or had snapshots that failed to make it in the pushed Snaps repository as more students neglected to push their Snaps repositories after the final project than in previous projects.

Table 1 shows the number of unique students our dataset contains snapshots for for each assignment.

Table 1: Count of unique students who have recorded snapshots for each CS 61B Spring 2021 assignment.

Assignment	Count
Project 1	1391
Lab 1	1359
Project 0	1334
Lab 3	1291
Lab 2	1266
Project 2	1208
Lab 7	1136
Lab 6	936
Lab 4	369
Lab 5	239
Lab 12	181
Project 3	88
Lab 13	20

Based on these findings and reasoning, we removed snapshots from assignments Lab 13, Project 3, Lab 12, Lab 4 and Lab 5 from our analysis.

Additionally, by manually inspecting the Snaps repositories resulting in the lowest and highest total working times computed (this calculation is detailed in the "Building a Working Time Metric" section below), we found that some students manually made commits to

their Snaps repositories or had large gaps in their snapshot history. As a result, we did not consider students with a total calculated working time of less than 10 hours in our analysis.

After cleaning the data based on manual inspection of Snaps repositories, the details of certain class assignments, and the grading policy chosen, we analyzed over 1.5 million (1,644,418) snapshots from 954 unique students.

4 METRICS

Given some snapshot containing a timestamp, a message listing files changes and code change data, we used a list of per-assignment file names to classify that snapshot by which assignment was being worked on when it was made. For example, in Project 0, students work on a file called Board.java, so any snapshots that contain edits to Board.java are considered a Project 0 commit.

Note that snapshots can belong to multiple assignments, as it is possible that a student changed files from multiple assignments in between IntelliJ "saves".

To formalize this, for each student i and each assignment k, we have an ordered set $S_{ik} = \{t_1, t_2, t_3, \ldots, t_n\}$, where t_i is the timestamp of the i'th snapshot taken. Specifically, t_1 is the timestamp of the earliest snapshot taken when student i was editing files for assignment k, and t_n is the timestamp of the final snapshot taken when student i was editing files for assignment k. Notice that for different students and assignments, the number of snapshots ($|S_{ik}|$) will not necessarily be the same, and could even be zero if a student skipped an assignment entirely.

4.1 Building a Working Time Metric

In order to compute an estimate of the amount of time a student spent working on an assignment, we used an approach employed in the previous Snaps report [10], and in other analyses of timestamped code snapshot data to estimate the amount of time a student spends in IntelliJ [9][14]. To convert a list of timestamps attributed to one assignment, we take a running sum of the gaps in between timestamps, only including timestamp gaps under a certain value in the sum.

Formally, we compute the time student i spent working on assignment k as

$$T_{ik} = \sum_{t_i \in S_{ik} \wedge i \neq 1} (t_i - t_{i-1}) \mathbb{1}[t_i - t_{i-1} < C]$$

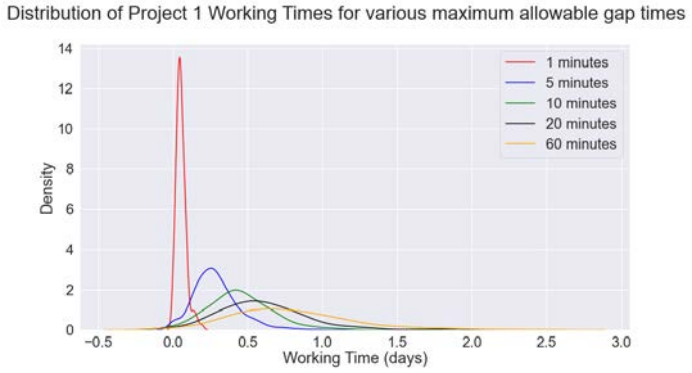
C is some constant representing the maximum time between snapshots that should still be added to the running sum. We will refer to C as the "maximum allowable gap time". The choice of C is nontrivial, as there are many reasons why a student may not be actively writing code in IntelliJ. A student may, for example, be watching a video walk through of a concept related to the assignment they are working on, using a debugger, or text messaging their friends. In the first two cases, we want to consider the time as a part of the working time, but not in the third case.

To illustrate, consider the following example. Suppose Snaps collects a snapshot for the Lab 1 assignment from a student at time 0. Snaps also collects snapshots for the Lab 1 assignment at time 15 seconds (15 seconds after the one at time 0), time 30 seconds, time 10 minutes, and then every 15 seconds after time 10 minutes until time 20 minutes is reached. Given this record for Lab 1, the working time calculated for this student would depend on the maximum allowable gap time. Using a maximum allowable gap time of 1 minute, we would calculate this student's working time on Lab 1 to be 10.5 minutes, as we would not include the 9.5 minute gap between time 30 seconds and time 10 minutes in the working time. Using a maximum allowable gap time of 10 minutes, we would calculate this student's working time on Lab 1 to be 20 minutes.

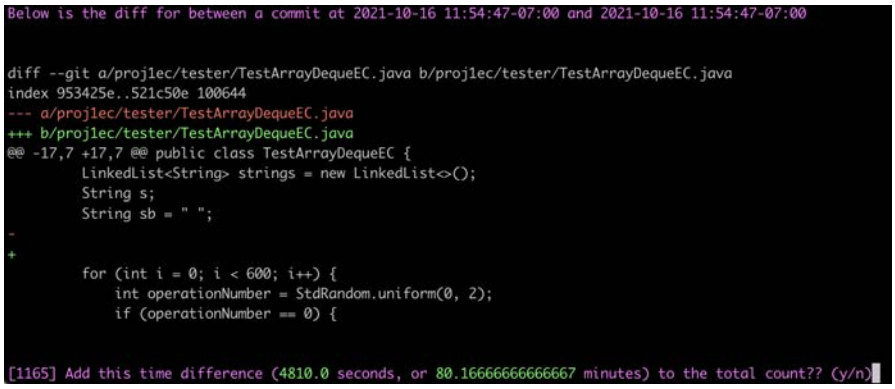
In previous work, the maximum allowable gap time chosen has differed: 10 minutes is used in Yan et. al. [14] and in the previous Snaps report [10], and 20 minutes is used as an example in Rodriguez-Rivera et. al. [9]. In Figure 1, we overlay plots of the estimated distribution of working times on Project 1 calculated in this way using different values for the maximum allowable gap time (ranging from 1 to 60 minutes).

The choice of maximum allowable gap time clearly influences the distribution. When using 1 minute, the mode working time is around .1 days, whereas when using 20 minutes, the mode is nearer to .6 days. In order to find a maximum allowable gap time to use when creating the working time metric for our analysis, we chose to dissect the actual code changes that made up the snapshots collected to get a "ground truth" approximation of working time.

To get the "ground truth" approximation, we built a tool to manually examine the code differences between subsequent snapshots and determine whether that gap time should be counted in the running total or not for



(a) Distribution of estimated working times on Project 1 for various maximum allowable gap times.



(b) The tool used to manually inspect differences between subsequent snapshots. The tool shows the changes each snapshot made with respect to the previous snapshot, and the user indicates whether or not that time gap should be added to the running total working time.

Figure 1: Determining a maximum allowable gap time for our working time metric.

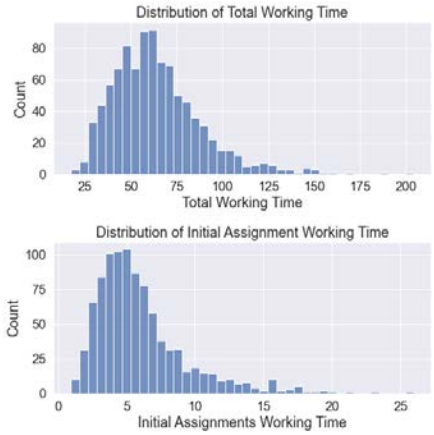
snapshots from Project 1 for 5 students. 2 labelers with experience taking CS61B examined these Project 1 snapshots. The factors used to make the decision whether to include a gap time or not was largely subjective, but based on the labeler's experience, the code differences between the snapshots, and the time elapsed. One case often caught by the labelers was snapshots that were clearly from a period of time when the student was debugging, consisting of 5-10 minute gaps and small, repetitive changes around the same few lines of code. A screenshot from this tool can be seen in Figure 1.

We determined that using a maximum allowable gap time of approximately 20 minutes resulted in calculated working times that were closest to the average of the "ground truth" approximations. From this observation, we decided to use 20 minutes as the maximum allowable gap time for all student work.

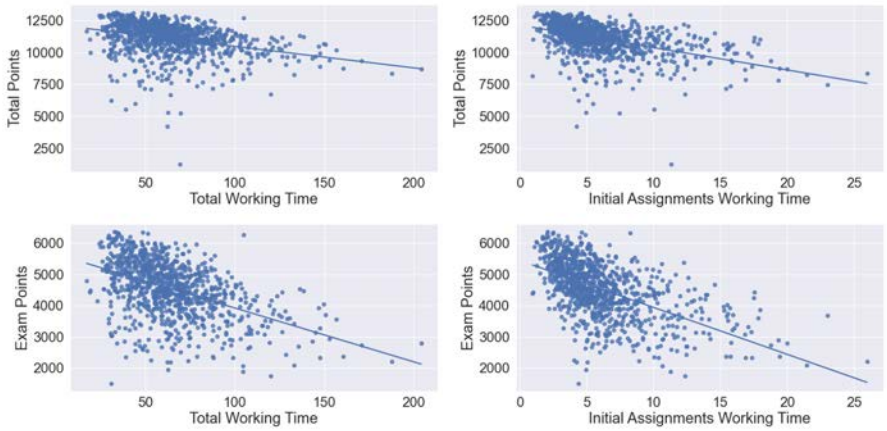
4.2 Working Time Results

We calculate the total working time of a student by summing the per-assignment working times. We additionally calculate the "initial assignment" working time of a student by summing the per-assignment working times only for the first three assignments of CS61B Spring 2021, Lab 1, Lab 2, and Project 0. Appendix A details why we chose these assignments in particular. If a student had no snapshots for a given assignment, when computing the sum, their working time was considered to be the average working time amongst all other students. Figure 2 depicts the distribution of total working times and initial assignment working times in hours in CS61B Spring 2021.

It appears that the distribution of working times is relatively normal, with a long right tail. A portion of outlier students spend considerably more time than their peers working on CS61B assignments. Next, we



(a) Distribution of total and initial assignment (Lab 1, Lab 2, and Project 0) working times in hours in CS61B Spring 2021. For total working time, the lower quartile was 47.38 hours, the median was 60.94 hours, and the upper quartile was 75.46 hours. For initial assignment (Lab 1, Lab 2, and Project 0) working time, the lower quartile was 3.74 hours, the median was 5.2 hours, and the upper quartile was 7.15 hours.



(b) Relationships between total working time and total course points (top left, $R^2 = 0.104$), total working time and exam points (bottom left, $R^2 = 0.226$), initial assignment working time and total points (top right, $R^2 = 0.212$), and initial assignment working time and exam points (bottom right, $R^2 = 0.321$).

Figure 2: Distribution of our working time metric, and associations between working time and course performance.

investigate the relationship between working time and performance. It is intuitive to believe that students who spend longer on assignments should do better, to a point. Those who spend too little time may not engage with the material enough, while spending too much time is likely an indication of struggling with the material in the course.

Figure 2 gives the relationship between working times and class performance. We measure class performance through total points in the class (out of 12800 possible points), as well performance on only the exams (out of 6400 possible points). Using total working time as a predictor, the coefficient of determination was $R^2 = 0.104$ for total points, and $R^2 = 0.226$ for only exam points. Using initial assignment working time as a predictor, the correlation coefficient was $R^2 = 0.212$ for total points, and $R^2 = 0.321$ for exams.

The scatterplots hint at an inverse relationship between working time and performance, though there are a number of outlying points in the bottom left corner,

students who performed poorly and spent very little time on the course, that decrease the strength of the association. The long right tail of working time is associated with a clear decrease in exam performance, while the average working time student appears in a noisy cloud.

It is evident that similar or stronger associations are seen just by using the first three assignments completed by CS61B students. We are unsure of the cause of this result, but hypothesize that working times on initial assignments are positively correlated with total working times, in addition to the lower likelihood of the Snaps plugin malfunctioning during the first few assignments, limiting the noise in the initial assignment data. We move onto building metrics from the Snaps dataset to uncover information about how students work on assignments beyond time spent.

4.3 Building Working Habits Metrics

In addition to the total amount of time a student spends on an assignment, we are also interested in the details of when students complete assignments in CS61B. Currently, CS61B course staff uses rough metrics such as checking the total number of submissions made to an assignment's autograder as the deadline approaches to understand when students are completing assignments. This approach only captures the behavior of the entire class as a whole, and fails to incorporate students who may be working on the assignment but have not yet made a submission to the autograder.

Using the timestamped snapshots, we define a metric that can be used as a measure of when students are completing assignments. The "lateness" metric is calculated as the average snapshot timestamp, minus the midpoint timestamp in between the assignment release date and the assignment due date. Using the formalization above, if assignment k has release date R_k and due date D_k, the "lateness" of student i on assignment k is given by:

$$L_{ik} = \frac{1}{n} \sum_{t \in S_{ik}} t - \left(\frac{D_k + R_k}{2} \right)$$

Intuitively, positive lateness values mean the student's average commit time is past the assignment midpoint, with more positive lateness indicating that more of the student's work is done after the assignment midpoint. Similarly, negative lateness values indicate the student's average commit time is before the assignment midpoint.

Subtracting the midpoint has the effect of centering the average timestamp around 0. If a student did all of the work for an assignment on the deadline (had an average commit time of D_k), their lateness would be equal to $\frac{D_k - R_k}{2}$, and if a student did all of the work for an assignment on the release date, their lateness would be equal to $-\frac{D_k - R_k}{2}$.

Lateness does not give all of the details an instructor may be interested in while considering when students complete assignments. A student who completed the entire assignment in one sitting at the midpoint between the release date and the due date, and a student who spent a few minutes working on the assignment every day in between the release date and the due date would both have very similar lateness scores (as the average commit time would be similar), but clearly these students completed this assignment in a very different manner.

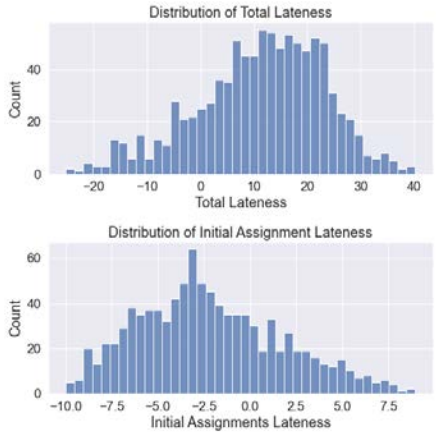
Again using the timestamped snapshots, we define a metric that can be used to differentiate the two cases above (and provide more information than the "lateness" metric alone). The "spread" metric is calculated as the standard deviation of the snapshot timestamps. Formally, the spread of student i on assignment k defining $\bar{t} = \frac{1}{n} \sum_{t \in S_{ik}} t$ is:

$$\sigma_{ik} = \frac{1}{n} \sqrt{\sum_{t \in S_{ik}} (t - \bar{t})^2}$$

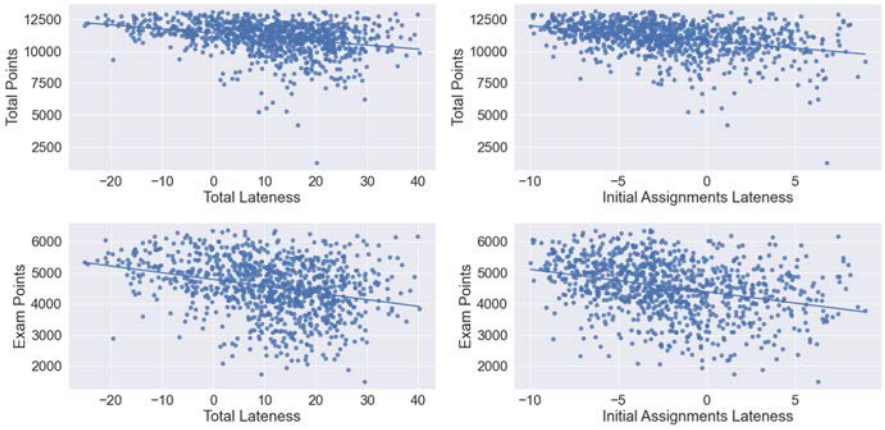
Intuitively, large spread indicates the student worked on the assignment over a longer period of time, and worked more steadily. Small (close to 0) spread would indicate the student did much or all of the work on the assignment in a short interval of time.

4.4 Working Habits Metrics Results

In order to aggregate the lateness and spread measures calculated per assignment, we sum the measures across all assignments, and across the initial assignments used in our previous analysis (Lab 1, Lab 2 and Project 0) to get a "Total Lateness/Spread" and a "Initial Assignments Lateness/Spread" measure. When calculating this aggregated metric, we replace missing values (students who did not have any timestamps for a given assignment) with the average lateness or spread for that assignment.



(a) Distributions of total lateness in days (lower quartile=3.99 days, median=12.4 days, upper quartile=19.84 days) and initial assignment lateness (lower quartile=-5.05 days, median=-2.63 days, upper quartile=0.37 days).



(b) Relationships between total lateness and total course points (top left, $R^2 = 0.088$), total lateness and exam points (bottom left, $R^2 = 0.084$), initial assignment lateness and total points (top right, $R^2 = 0.127$), and initial assignment lateness and exam points (bottom right, $R^2 = 0.100$).

Figure 3: Distribution of our lateness metric, and associations between lateness and course performance.

Figure 3 gives the distribution of total lateness for all assignments and total lateness for only initial assignments for all of the students in the cleaned Snaps dataset. We omit plotting distributions for total and initial assignment spread, as this metric is less interpretable.

The total lateness distribution is relatively normal, with a heavier left tail. There appears to be a portion of students who consistently start assignments early, which leads naturally into investigating associations between lateness and performance. The distribution for initial assignment average lateness is considerably shifted left from the distribution for total average lateness, which we hypothesize is due to a combination of student work ethic attrition, higher variance in behavior over fewer assignments, and assignments increasing in difficulty and length as the semester goes on.

Figure 3 additionally gives relationships between lateness and course performance. The general trend, though weak in association, is a negative relationship between our lateness metric and performance in the course. The association using the lateness metric calculated from all assignments has similar strength to the association seen using our working time metric,

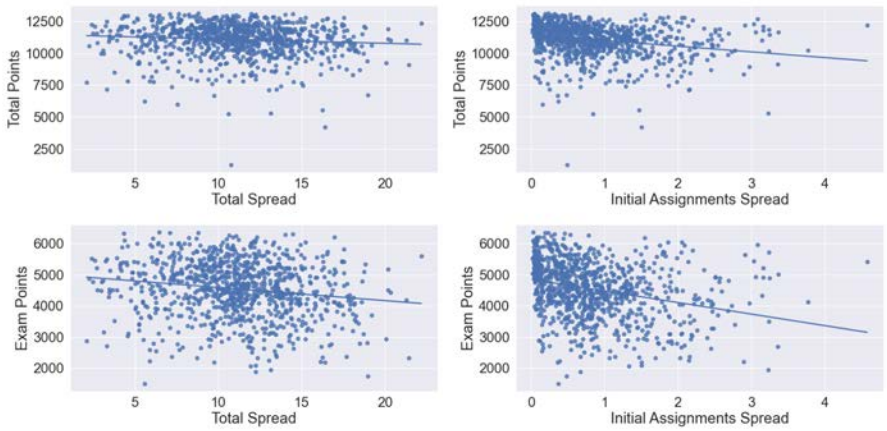


Figure 4: Relationships between total spread and total course points (top left, $R^2 = 0.008$), total spread and exam points (bottom left, $R^2 = 0.027$), initial assignment spread and total points (top right, $R^2 = 0.057$), and initial assignment spread and exam points (bottom right, $R^2 = 0.081$).

while the metric calculated using only the initial assignments is less strong than that seen using the initial assignment working time metric. Additionally, the association when using Exam Points as the dependent variable is less strong than the association seen using our working time metric.

We hypothesize that these results stem from assignment working habits not being as informative about exam performance as assignment working habits are likely not tied to theoretical understanding. We do see a similar trend that using only the data from the initial assignments gives a stronger association, which we hope indicates that only initial assignment data may be usable as a good predictor of success.

Figure 4 gives relationships between our spread metric and course performance. The association between the spread metric and performance in the course (both using Total Points and Exam points as the dependent variable) is essentially nonexistent. We believe that the spread metric may be more useful for predicting success in conjunction with the lateness (and working time) metrics. We turn to exploring the performance of predictive models using these metrics as features.

5 PREDICTING SUCCESS

Above, we explored the association between student performance as measured by total class points and exam points, and aggregated versions of our working time, lateness and spread metrics. Now, we move onto utilizing all of the per-assignment metrics to fit models for prediction of these performance outcomes. We start with a baseline model that only uses a student's CS61A (CS 1 at UC Berkeley) grade, as this was determined to be a useful predictor of course performance in previous work [6]. We then incorporate our per-assignment metrics for various subsets of assignments into these models to investigate any improvement in model accuracy, with special attention to early subsets of assignments.

Students' CS61A grade are self-reported on the Pre-Semester survey. In the survey, students were asked "If you took CS 61A at Berkeley, what grade did you get?". The possible responses were the letter grades A-F (with + and - for A-C as options), P or NP for the pass/fail grading option, "I did not take CS61A at Berkeley" and "Decline to state". As the Pre-Semester survey was voluntary, use a subset of our dataset containing only students who responded to the survey (N=856).

As a baseline, we one-hot encode the CS61A grade data, and use a LASSO linear regression model [11], which regularizes coefficients and tends to give weight of 0 to features that only result in overfitting to noise. While overfitting is unlikely with so few features, the LASSO model is used so that a comparison can be made when incorporating our metrics, as this will increase the amount of features. Both total points and exam points are used as the prediction targets. We report the average 5-fold cross validation coefficient of determination

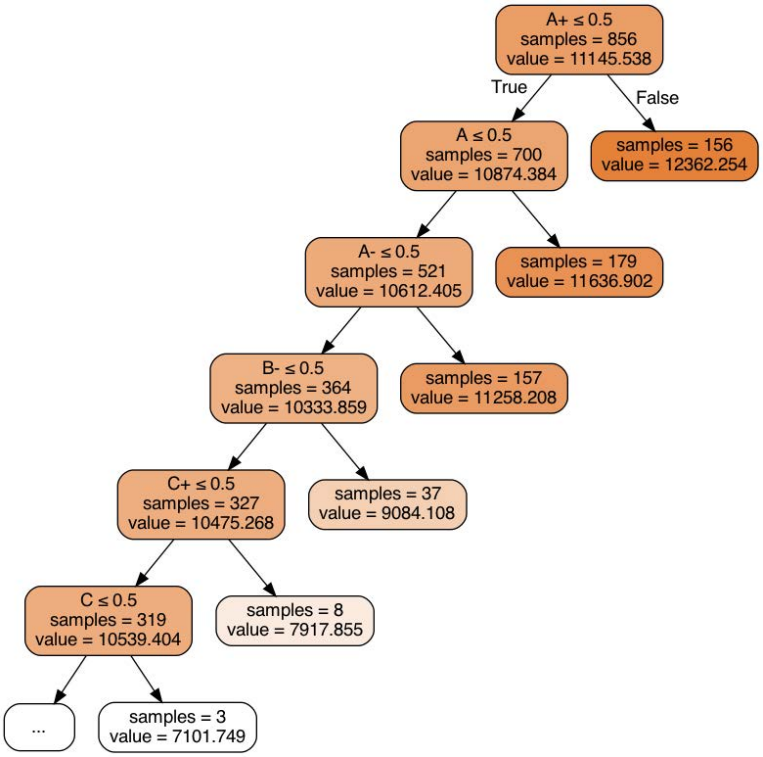


Figure 5: A portion of the regression decision tree fit using only self-reported CS61A grade to predict a student's total points in CS61B. Grades were one hot encoded, so a value of 1 corresponds to having earned that grade in CS61A. The "value" of each node corresponds to the average CS61B grade earned by datapoints in that node, and the "value" in a leaf node is the prediction returned by the model for datapoints in that node.

(R^2) score as a measure of the model's accuracy. We use the cross validation score to evaluate our models as we are interested in their ability to predict unseen data, as might be the case in a future semester. As a result, an apples to apples comparison cannot be made between the predictive models' R^2 scores and those of the aggregated working time, lateness and spread metrics as the scores reported above resulted from fitting on the entirety of the dataset.

Using only the CS61A grade data, an average 5-fold cross validation R^2 value of 0.557 is achieved when predicting total points, and a value of 0.610 is achieved when predicting exam points. This result agrees with the findings of previous work, as the coefficient of determination values from CS61A grade data are considerably higher than any from our aggregated metrics (working time, lateness, and spread).

Figure 5 visualizes a portion of a regression decision tree fit on the one-hot encoded CS61A grade data. The decision tree algorithm will just form a spindly tree that predicts the average value in each category for other values in that category. As expected, students who earned higher grades in CS61A are predicted to earn a higher grade in CS61B. Particularly, students who earn an A+ in CS61A are predicted to receive an A in CS61B, while students who earn a C in CS61A are predicted to earn a C- in CS61B.

Given baseline R^2 scores from using only CS61A grade data, we consider models that incorporate both CS61A grade and our metrics to predict course performance. We fit models using assignments up to and including Project 0 (2 weeks into the course, also referred to as "Initial Assignments"), up to and including Project 1 (5 weeks into the course), and all remaining assignments used in the dataset (11 weeks into the course). As now some features are not just binary values, we standardize each feature before fitting LASSO linear regression models. Table 2 gives the average 5-fold cross

Assignments Used	Total Points R^2	Exam Points R^2
CS61A Grade	0.557	0.610
CS61A Grade and Initial Assignments	0.585	0.656
CS61A Grade and Assignments Through Project 1	0.585	0.670
CS61A Grade and All Assignments	0.588	0.666

Table 2: Average 5-fold cross validation R^2 scores of LASSO linear regression models for various sets of features that include CS61A (CS 1) grades using total points and exam points in CS61B as outcomes.

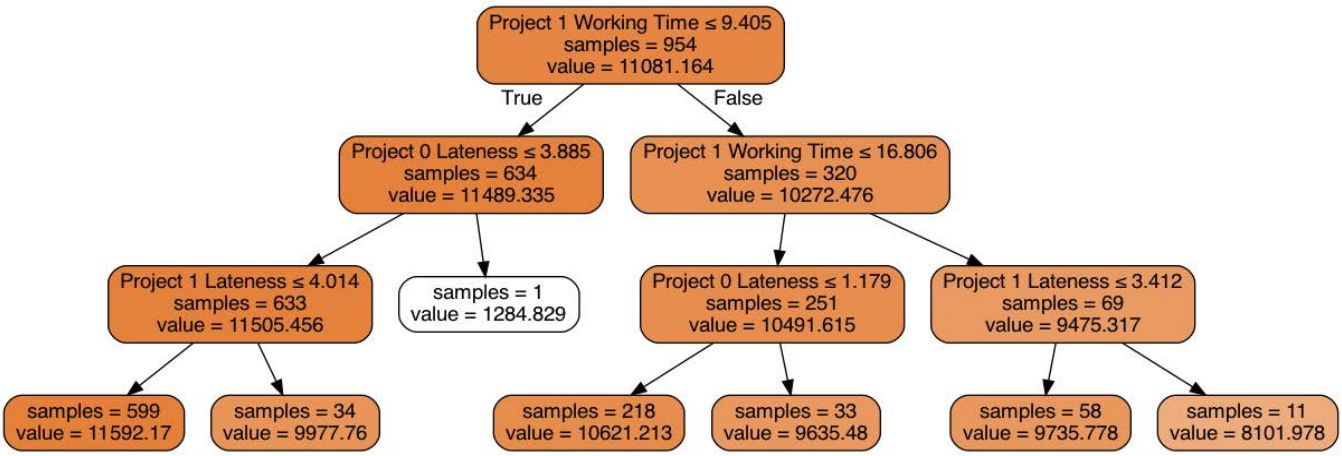


Figure 6: A visualization of a regression decision tree using working time, lateness, and spread metrics from all CS 61B assignments to predict total points ($R^2 = 0.185$). The "value" of each node corresponds to the average CS61B grade earned by datapoints in that node, and the "value" in a leaf node is the prediction returned by the model for datapoints in that node.

validation coefficient of determination (R^2) score for fitting the LASSO linear model using different sets of features to Total Points and Exam Points outcomes.

As observed when comparing relationships between each of our metrics individually with student points, our metrics in addition to CS61A grade data are able to predict exam scores more accurately than the total points earned in the class. This is counterintuitive as the Snaps data comes from only the assignments, but we believe this result is related to exam scores being a less noisy measure of student performance, as well as the data cleaning process in which we replaced missing exam scores with the average exam score.

While using our metrics in addition to the CS61A grade data does not show spectacular increase in coefficient of determination score, we are encouraged by the results as including our metrics as features gives an average cross validation R^2 increase of about 3% when predicting total points and about 6% when predicting exam points. Furthermore, we see a similar increase when only including assignments from the first two weeks of the course.

In order to understand more about what student behaviors result in high or low predicted performance as measured by our metrics, we fit a regression decision tree using data from all of the assignments to predict total score. We used a cross validation grid search to select hyperparameters for the decision tree in order to prevent overfitting while still being complex enough to predict performance. The search resulted in a decision tree that minimizes the squared error loss and has a maximum depth of 3. Figure 6 gives a visualization of the decision tree, which resulted in an average cross validation coefficient of determination score of 0.185.

Following the branch that leads to the highest predicted score (11592.17, an A-), we find that the model gives this prediction to students who:

- Spend less than or equal to 9.405 hours on Project 1.
- Have an average Project 0 commit time that is less than or equal to 3.885 days after the midpoint.
- Have an average Project 1 commit time that is less than or equal to 4.014 days after the midpoint.

Following the branch that leads to the lowest predicted score that is not a single datapoint (8101.978, a C+), we see that this prediction is assigned to students who:

- Spend more than 16.806 hours on Project 1.
- Have an average Project 1 commit time that is more than 3.412 days after the midpoint.

The "midpoint" refers to the halfway point between the release date and due date of an assignment.

The takeaways from this decision tree model are limited given its low average cross validation score compared to the LASSO models, as well as the randomness involved in the order of splits that are considered. That said, it is clear that the combination of lateness and working time metrics result in informative splits (without one clearly dominating the other), while the spread metric is rarely chosen to split a node. Furthermore, some of the same associations from the single-metric evaluation are seen in the paths to the lowest and highest predicted scores in the decision tree: spending too long on an assignment, or starting too late results in predictions for performing worse.

6 DISCUSSION AND FUTURE WORK

6.1 Instructor Intuitions

One motivation for building metrics from the Snaps dataset was to investigate the validity of the rough intuitions instructors use to evaluate course performance and make policy decisions.

When building the working time metric, we were interested in determining whether or not spending more time on assignments resulted in better performance, as instructors tend to see spending too much time as an indicator of struggling with concepts, and too little time as being detrimental to understanding. When building the lateness and spread metrics, we desired to understand if the student who starts early and works steadily really performs as well as instructors would expect them to, or if the student who finishes the assignment at the last possible moment actually is doing themself a disservice.

The rough negative association between our working time metric and class performance does agree with instructor intuition in that students who spend longer on both initial and all assignments are more likely to perform poorly on exams and receive a lower letter grade. As the negative association also shows that students who complete assignments very quickly are more likely to perform better, this analysis does not give good information regarding students who finish assignments so fast that they miss out on the learning objectives of the assignment. It is possible that fitting a curve with a term of degree two to the working time and performance scatterplot could give more information about this intuition. Finally, there are many additional factors at play, so there is no basis whatsoever to imply that the time a student spends working on an assignment causes a change in their score.

Again, the rough negative association between our lateness metric and class performance aligns with intuition. Students who start earlier are more likely to earn more points on exams and in the course. The effect of lateness is slightly stronger than that of working time with respect to total points, but weaker with respect to exam points. One possible interpretation is that this result is due to assignment completion behaviors being more independent of exam studying behaviors while working time is closer tied to conceptual understanding, though further work is necessary to confirm this claim.

The association between our spread metric and class performance is negligible. We continue to believe that there is more information to be had from the times at which students are working on assignments than only the average (our lateness metric).

In order to make stronger statements about these instructor intuitions, we are interested in surveys of instructors to confirm claims of what intuitions are held and used. As UC Berkeley Computer Science courses regularly gather self-reported data, incorporating data such as self-reported working time, starting time and difficulty level may give more insight into the reliability of these intuitions.

Additionally, we believe that a more robust method of computing a working time that relies on the contents of student code snapshots, and a tool less prone to error

could improve the strength of the performance-working time association. The Snaps tool could be improved in a number of ways depending on the intended usage. If one wanted only to use the timestamps of the snapshots as in this report, the tool could be modified to only capture timestamps, which would lower the complexity of setup required and likely reduce the amount of noise in the data. If the code differences are desired, one could standardize the amount of time between snapshots rather than relying on IntelliJ autosave events, and increase the frequency between required "checkpoints" in which students push their Snaps repositories to limit malfunctions and decrease variance in the data. Finally, one could improve the way in which snapshots are associated with particular assignments by using more information than the filename, such as the directory the student made changes in (in CS61B, each assignment gets its own directory).

6.2 Early Prediction

A second motivation for building metrics from the Snaps dataset was to determine the ability of models to predict student success using data from only assignments completed early on in the semester. With accurate early prediction models, instructors could have the ability to identify struggling students and update policy decisions despite large class sizes.

We found that relatively simple linear models were able to predict students' total exam scores and final grades with limited accuracy based on our metrics, though the models were more accurate at predicting total exam scores than final grades. By fitting a decision tree model, we found anecdotal evidence that the lateness and working time metrics were most effective for predicting performance.

As suggested by previous research, we found that using grade data from CS61A (CS1 at Berkeley) resulted in substantially more accurate models. When combined with data from CS61B assignments from the first 2 weeks of the course, we saw an increase in accuracy, which indicates that our metrics gathered from these initial assignments can result in early prediction of final grade and total exam scores.

An instructor could potentially implement an intervention for students who, per an early prediction model based on Snaps metrics and previous course grade, are more likely to perform poorly in the course. It would be telling to gather data on how student performance changed as a result of this intervention as compared to performance based on this dataset. This intervention should be vetted and well thought out to control for the risk of bias associated with using the predictions of a statistical model in a human setting, as well as the damage an incorrect prediction might have.

While using our metrics calculated from snapshot timestamps as features does improve accuracy from the naive CS61A grade model, we believe that there is rich information available in the code contents of the snapshots. Careful manual inspection, in addition to static code analysis and more complex machine learning techniques may result in better metrics, as well as stronger predictions.

Finally, we acknowledge that the use of point totals as a proxy for student success may not be fully sound. There are other metrics for success that may be more beneficial for students, instructors, and model accuracy. One potentially informative outcome is performance on the first or second midterm exam alone (rather than the total performance on exams), which would also allow for investigating what results in improved (or worse) performance comparing early in the course to the end of the course. We would also be interested in considering self-reported satisfaction level or goal achievement as a different way of measuring success.

7 ACKNOWLEDGEMENTS

I am lucky to have found peers and mentors who provided me with invaluable advice and support as a student, teaching assistant, and while working on this report. Specifically, I'd like to thank Itai Smith, Prof. Josh Hug and Daniel Edrisian for their contributions and guidance. Additionally, this report would not have been possible without the hard work and dedication of the CS 61B Course Staff.

REFERENCES

[1] UC Berkeley Course catalog. URL https://classes.berkeley.edu/.

[2] About: Small and fast. URL https://git-scm.com/about.

[3] About: CS 61B Spring 2021, 2021. URL https://sp21.datastructur.es/about.html.

[4] K. Arakawa, Q. Hao, W. Deneke, I. Cowan, S. Wolfman, and A. Peterson. Early identification of student struggles at the topic level using context-agnostic features. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science*

Education V. 1, SIGCSE 2022, page 147–153, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450390705. doi: 10.1145/3478431.3499298. URL https://doi.org/10.1145/3478431.3499298.

[5] L. Beck, P. Kraft, and A. W. Chizhik. Predicting student success in cs2: A study of cs1 exam questions. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2022, page 140–146, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450390705. doi: 10.1145/3478431.3499276. URL https://doi.org/10.1145/3478431.3499276.

[6] A. Guo. Analysis of factors and interventions relating to student performance in CS1 and CS2. Master's thesis, EECS Department, University of California, Berkeley, May 2020. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-22.html.

[7] JetBrains. Save and revert changes: Intellij IDEA, Nov 2021. URL https://www.jetbrains.com/help/idea/saving-and-reverting-changes.html.

[8] N. A. of Sciences Engineering and Medicine. *Assessing and Responding to the Growth of Computer Science Undergraduate Enrollments*. The National Academies Press, Washington, DC, 2018.

[9] G. Rodriguez-Rivera, J. Turkstra, J. Buckmaster, K. LeClainche, S. Montgomery, W. Reed, R. Sullivan, and J. Lee. Tracking large class projects in real-time using fine-grained source control. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2022, page 565–570, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450390705. doi: 10.1145/3478431.3499389. URL https://doi.org/10.1145/3478431.3499389.

[10] I. Smith. Snaps: A tool for understanding students in large computer science classes. Master's thesis, EECS Department, University of California, Berkeley, May 2021. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-118.html.

[11] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 00359246. URL http://www.jstor.org/stable/2346178.

[12] B. C. Wilson and S. Shrock. Contributing to success in an introductory computer science course: A study of twelve factors. *SIGCSE Bull.*, 33(1):184–188, feb 2001. ISSN 0097-8418. doi: 10.1145/366413.364581. URL https://doi.org/10.1145/366413.364581.

[13] L. Yan, N. McKeown, M. Sahami, and C. Piech. Tmoss: Using intermediate assignment work to understand excessive collaboration in large classes. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, page 110–115, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450351034. doi: 10.1145/3159450.3159490. URL https://doi.org/10.1145/3159450.3159490.

[14] L. Yan, A. Hu, and C. Piech. Pensieve: Feedback on coding process for novices. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 253–259, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450358903. doi: 10.1145/3287324.3287483. URL https://doi.org/10.1145/3287324.3287483.

A APPENDIX A: CHOOSING INITIAL ASSIGNMENTS

Predicting student success at an early stage in a course allows instructors to both implement interventions that target students predicted to do poorly as well as evaluate syllabus decisions that can impact the predictors of success.

Using the Snaps data, we investigate relationships between student work habits and performance in the course using per-assignment metrics computed from the timestamped snapshots. To get a big picture understanding of the interaction between these metrics and performance in the course as measured by exam points and total points, we aggregate the per-assignment metrics. When evaluating early success prediction, we are interested in finding a subset of assignments delivered early in the semester that are still predictive of total points and exam points.

To choose this subset of assignments, we line the assignments up in chronological order based on the due date. Then, for each assignment, we fit linear regressions using the working time calculated for that assignment and all previous assignments as features on both total points in the course and total exam points. Figure 7 plots the "score" (cross validation coefficient of determination) of each model. Using 5-fold cross validation allows us to control for overfitting, as the coefficient of determination on the entire dataset would always increase as a result of adding a new feature.

Based on this analysis, it appears that Project 0 had the most predictive power (or explained a higher proportion of the variance in exam and total scores), as adding metrics calculated from this assignment as a features resulted in the largest increase in the cross validation coefficient of determination of the linear model. Given that Project 0 is due two weeks into the semester, we chose to use Lab 1, Lab 2 and Project 0 as the "initial assignments" when aggregating metrics and investigating associations.

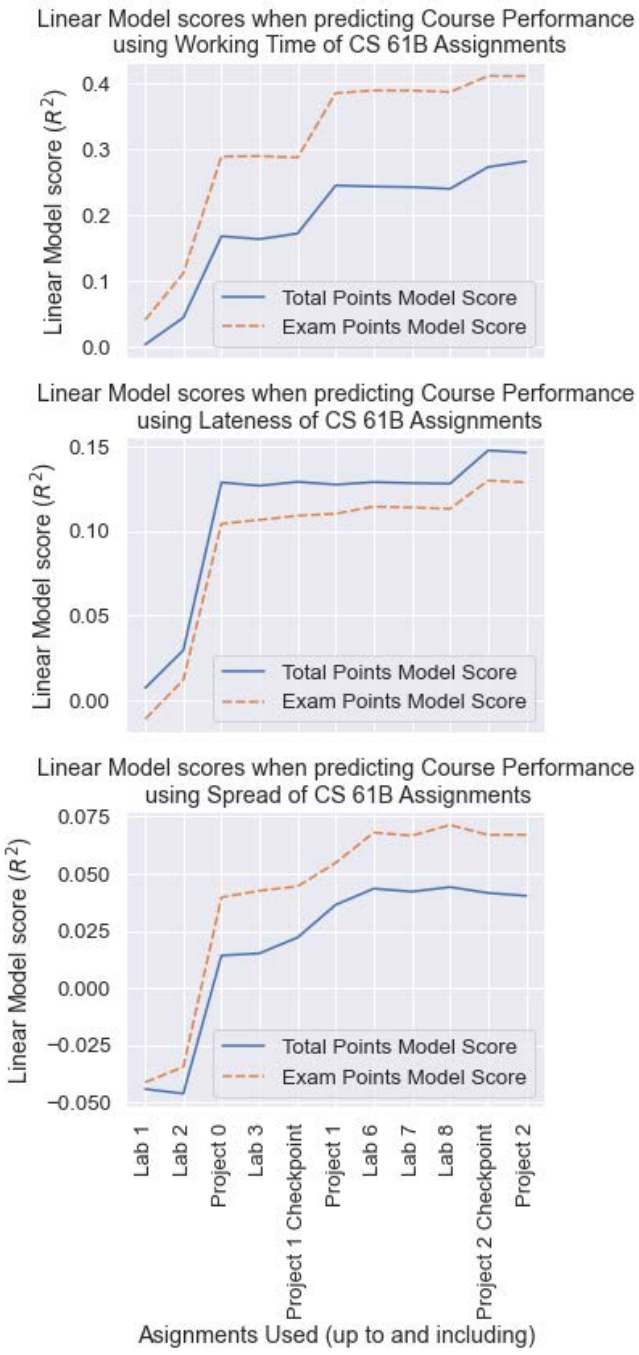


Figure 7: 5-fold cross validation scores (coefficient of determination) for linear regressions fit using working times, lateness, and spread of CS 61B assignments based on chronological order of due date. The linear regression corresponding to a particular assignment uses the metric from that assignment and any assignment due before it as a feature.