HappyFace: Identifying and Predicting Frustrating Obstacles for Learning Programming at Scale

Ian Drosos
NC State University
Raleigh, NC, USA
izdrosos@ncsu.edu

Philip J. Guo
UC San Diego
La Jolla, CA, USA
pg@ucsd.edu

Chris Parnin
NC State University
Raleigh, NC, USA
cjparnin@ncsu.edu

***Abstract*—Unnecessary obstacles limit learning in cognitively-complex domains such as computer programming. With a lack of appropriate feedback mechanisms, novice programmers can experience frustration and disengage from the learning experience. In large-scale educational settings, the struggles of learners are often invisible to the learning infrastructure and learners have limited ability to seek help. In this paper, we perform a large-scale collection of code snippets from an online learn-to-code platform, Python Tutor, and collect a frustration rating through a light-weight learner feedback mechanism. We then devise a technique that can automatically identify sources of frustration based on participants labeling their frustration levels. We found 3 factors that best predicted novice programmers' frustration state: syntax errors, using niche language features, and understanding code with high complexity. Additionally, we found evidence that we could predict sources of frustration. Based on these results, we believe an embedded feedback mechanism can lead to future intervention systems.**

I. Introduction

Frustrating obstacles when coding [1] can cause a learner to disconnect from the learning process and can even lower their self efficacy. In massive open online courses (MOOCs) and learn-to-code websites such as Python Tutor [2], frustration can come from course difficulty, lack of support, lack of computer literacy or self-teaching skills, and failed expectations [3]. When students cannot overcome these obstacles, they often drop out [4], with less than 15% of learners completing MOOCs [5]. There is a need to discover these frustrations felt by learners so that tools which provide interventions to assist learners can be more effective and targeted.

Learners from around the world are now writing, running, and debugging code from introductory courses or may be trying to debug more advanced coding assignments. While coding, learners inevitably run into frustrating events ranging from syntax errors to misunderstanding features of the programming language. While these frustrating events do not always have a negative impact on the learner [6], frustrations caused by cognitive challenges can affect user retention [7] and limit the effectiveness of the platforms the learners are using. Specifically, in large-scale learning systems, traditional mechanisms for overcoming learning difficulties, such as peer feedback and tutor instruction, do not scale well. This results in instructors having limited context about the problems being solved, knowledge about the current mastery levels of a learner, and insight into a learner's potential misconceptions.

Several platforms have tried to help novice programmers at scale. For example, HelpMeOut suggests solutions to errors it collected from a crowd of programmers who had a similar error in order to help inexperienced programmers understand and correct errors [8]. While crowd-based strategies can be effective for resolving certain types of obstacles, maintaining incentives and quality of crowd-based systems can be difficult, especially for rapidly evolving technologies and for obstacles beyond interpreting an error message. Intelligent tutoring systems have been used to automatically assist learners by providing adaptive and individualized feedback about a learner's work. In particular, JavaTutor has been used to predict how a learner's affect (i.e., emotion) changes in response to questions from the automated tutor [9]. Unfortunately, obtaining affect data required for such prediction requires the use of physical sensors. For MOOCs, the population of learners is large and diverse, so that it would be ideal to not rely on each student having a webcam, Kinect, or any other sensor properly set up.

In this paper we present HappyFace, a system that allows a novice programmer to annotate their learning experience. HappyFace then automatically infers what type of learning obstacle they may be encountering. Even with low participation rates common in MOOC-like online settings, HappyFace can automatically discover types of frustrating experiences and predict categories of learner frustration. Our long-term goal is to support data-driven discovery of frustrating aspects of programming languages, assignments, and tools in a lightweight and unobstrusive manner. HappyFace has four components:

- **Affect Survey**. Embedded in a host system, HappyFace collects the current affect of a learner when the learner selects the face icon that best represents their current mood.
- **Feature Extraction**. HappyFace automatically analyzes the code the learner was working on when they reported their current affect. This step is done via static program analysis through the parsing of abstract syntax trees to extract features, and stylometric analysis through the extraction of features relating to the style of the code.
- **Correlation**. HappyFace correlates the extracted features with the user's affect vote to discover frustrating features.
- **Prediction**. HappyFace predicts the type of learning obstacle a learner is facing when they report their affect.

978-1-5386-0443-4/17/$31.00

In a 2.5-month online study, we found that many learners were willing to provide feedback about their current emotional state during a learning experience. We integrated HappyFace into an existing learn-to-code website called Python Tutor [2]. Python Tutor lets users to write code inside their browser and then visualize each execution step of their code, allowing the user to better understand what is going on inside their code. 2,385 Python users elected to use HappyFace during their coding session, which represented 2.35% of all Python Tutor users during the time span of the study. From learner annotations we found that syntax errors, features that increase code complexity, and niche language features, such as the Python Slice Operator and Globals, correlated with frustration in learners. An odds ratio analysis found that syntax and indentation errors doubled the odds of learner frustration, and each occurrence of boolean comparators, library imports, and use of Globals increased the odds of frustration by 1.5 to 1.7 times. In a follow-up study, we found that annotations could be used to help identify and predict frustrating experiences. Using logistic regression, we predicted when learners did not understand their code with 80% precision.

Despite HappyFace's affect survey being a simple 5-choice survey (Figure 2), the data we collected yielded an abundance of useful information on learner frustration without using more complex or invasive methods of data collection such as physical sensors. We were able to use this data to discover features causing frustration with only lightweight static program analysis. Based on these results, we believe our approach can be used to support future learning interventions at scale.

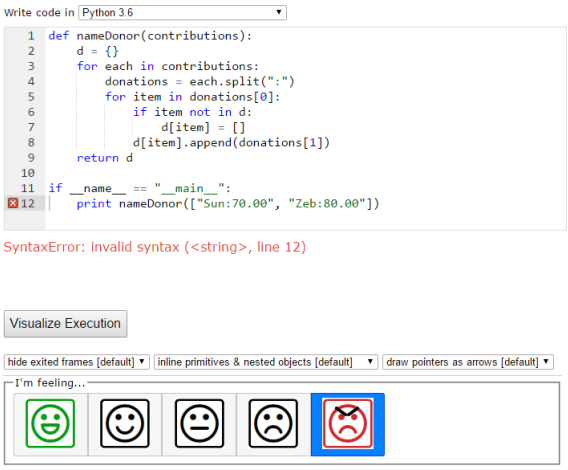

Fig. 1: Code example from a frustrated learner on Python Tutor with the HappyFace 5-choice affect survey shown below the code editor.

II. Motivating Example

For example, take a professor who is teaching an introductory programming MOOC and has just given the 1000+ students in the course a new programming assignment involving splitting strings and printing results. Since this is an introductory course, many of these students are new to programming and thus inevitably run into various issues completing the assignment. In previous semesters, the professor had to ask the TAs to obtain sources of frustration felt by the students and causes of errors in student code. The TAs attempted to obtain this information through interviews of the students via forum threads, but this requires a lot of manual effort. Also, while the TAs are able to find some categories of frustration, most of the students do not yet have the knowledge or experience to accurately convey the root causes of problems they faced.

Instead, the professor has deployed HappyFace this semester so students can report exactly when they are frustrated during an assignment. One student, Ada, is using Python Tutor to code up the assignment but when she executes her program she receives a SyntaxError (Fig 1). Ada grows frustrated as she does not understand why she is getting this error since she is not aware of the new syntactic rules in version 3 of Python.

When she encounters this error, Ada clicks a face icon that represents frustration on the HappyFace survey, located under the code editor. HappyFace can then analyze her code and report valuable information to the professor that shows some of the students are frustrated when dealing with SyntaxErrors. The professor can now adjust the lecture to include strategies to solving this error, alleviating any unnecessary frustration that may be felt by students like Ada attempting similar work.

III. The HappyFace System

Fig. 2: HappyFace affect survey with 5 choice buttons

HappyFace allows a learner to annotate their learning experience. We embed a survey to analyze these annotations by creating an abstract syntax tree (AST) from code snippets and process the code for language features and other stylometric features, such as types of whitespace, code complexity, and the standard deviation of line length. These scores are run through randomized logistic regression (RLR) for feature selection to identify potentially frustrating learning experiences. RLR finds features that are often selected over several runs of randomized "subsampling of the training data and fitting a L1-penalized LogisticRegression model" [10]. These frequently selected features can be considered good candidates for correlation with frustration. HappyFace is then extended to allow for participants to select additional reasons for frustration. Using our code metrics, we then predict the problem categories a code snippet will fall into.

A. Design Goals

To define HappyFace's design goals, we examined multiple feedback surveys in use today. Further, we tested a few prototype designs to assess which style of survey implementation would elicit the most responses from online learners. Lastly, in order to gather and analyze data provided by learners,

HappyFace must be adopted by learning frameworks. Based on these observations, we adhered to these design goals:

Brief and Minimalist: To encourage participation, the design must be brief and minimalist. If the survey clashes with the learning platform it is embedded in, that will serve as an unwanted distraction to the learning process. Thus, HappyFace is designed to be as brief as possible so a learner can use the platform as intended. HappyFace is also designed to be minimal. It is an inline survey that takes only the amount of visual space it requires. When a learner decides to cast a frustration vote, any further parts of the survey expand to allow the learner to provide more information and then collapse once a learner has finished answering the survey. This allows HappyFace to provide a feedback tool that minimally disrupts the appearance of the learning platform it is embedded in and allows learners to quickly return to their learning workflow.

The first prototype of HappyFace included a slider that manipulated the color and smile of a smiley face above the slider. When the user neared the maximum frustration rating the face became red and had a frown, when the user neared the minimum frustration rating the face became green and had a smile. This first prototype also had a free-form text area to allow the learner to explain why they were or were not frustrated in their own words. Once the frustration score was selected, the user could then press a button to submit their frustration score, frustration reason, and code snippet. This prototype survey was hosted on Python Tutor for a week but did not receive a sufficient amount of responses. We attribute this to the fact that it took multiple steps to answer the survey, potentially adding additional frustration to an already frustrated learner. Thus, we sought to redesign the interface.

To implement this design goal, we drew inspiration from the work of Bieri et al. on the self-assessment of pain severity [11]. The authors asked children to look at a set of faces depicting different levels of pain and order them from no pain to the most pain. The aim of the authors was to create a scale with "minimal cognitive demands". The authors found that their face-based pain scale required little direction of the child as they are simply told to point to the face that matches the amount of pain they feel. The quick and simple face-based pain scale showed that it could be used reliably and validly for the self-reporting of pain in children.

Thus, the HappyFace survey became a simple five button design that took inspiration from the pain scales used by medical professionals (Fig 2). The five buttons had faces with a range of emotions, from happy to angry. The only instruction provided to the learner is text saying "I'm feeling...", implying to the learner they should select the face that best represents their current feeling. Self-reporting of frustration on HappyFace is also very quick: a one-button click on the face that most represents the learner's frustration is all that is needed from the learner. Once a learner clicks a face, HappyFace collects the rest of the relevant data automatically.

Problem and Language Agnostic: To ease adoption, the design of the system must be able to accommodate the many different programming problems as well as programming languages that a learning platform may support. For example, several approaches for automated feedback of student assignments require that the problem be known ahead of time [9] and that it is accompanied by set of test cases [12]. Further, these techniques require heavier analysis, such as symbolic execution, which can work appropriately on a few student assignments, but may not scale well when needing to do an analysis of millions of code snippets.

To implement this design goal, we drew inspiration from the work of Caliskan-Islam et al. on code stylometry [13]. They used submissions to a Google Code Jam as a dataset, extracted stylometric features from the code, and attempted to de-anonymize programmers based on these features. Machine learning methods were run against the extracted feature sets and found that the author of a piece of source code could be correctly attributed at high accuracy (>90%). This result gave us confidence that properties extracted from the AST of the code could provide sufficient features without a more complex representation of problem state.

Thus, the analysis component of HappyFace was based on light-weight static analysis of the AST, using the same features as Caliskan-Islam et al. [13]. For example, some attributes, such as very long and complex lines of code, could contribute to frustration of the learner. This makes it important to detect this "Long Method" smell, as it decreases the ability of the learner to understand the given code [14]. The structure of most languages contain similar features like if statements, loops, and functions. Further, stylometric features, such as average line length, are language-agnostic. Because of this, HappyFace can be implemented for almost any language.

TABLE I: Extracted Features

Feature	Notes
Frustration Score	Given by user
AST Nodes	The Python standard library "ast" [15] [16] detailing the abstract syntax grammar of the code. Each node occurrence is collected
Stylometric Features	Occurrences of reserved Python keywords (keyword and builtin [17] [18] libraries), line count, average length of lines, standard deviation and variance of line length, white space count, types of white space used, underscore usage, comment count, count of lines starting with comment or tab character, empty lines. Derived from [13] Tables 2, 3, and 4
Wordgrams	Wordgrams extracted using regex to detect commonly used words in the code
Full Details	For more detailed information on features see https://github.com/wddlz/HappyFaceInfo

B. Implementation

1) Affect Survey: HappyFace has an inline survey that can be embedded into a system to collect frustration data from its users. The current prototype of HappyFace was implemented on PythonTutor.com [19], a learn-to-code website. (Note that HappyFace can be embedded into any learn-to-code website or IDE.) A user can use the HappyFace survey during their workflow on the website. In an extension of the affect survey, the user is also presented with a choice of tags they can select to better describe their vote in the form of a frustration class.

TABLE II: Selection categories for annotating learner affect

Category	Description
Happy Categories	"I fixed an error", "My output is right", "My code works", "I understand this code", "Other [user input required]"
Frustrated Categories	"I get an error", "My output is wrong", "Some code is not running", "I don't understand this code", "My variable has the wrong value", "My loop doesn't iterate correctly", "If statement does not work", "Other [user input required]"

2) Data Collection: HappyFace was embedded on Python Tutor's visualize page under the code editor and "Visualize Execution" button (Fig 1). While users of the site were programming on this page they could select a face that most represents their current affect. Once a face was selected, HappyFace collects the code the user has written along with the affect rating and stores the vote in a database for analysis. The survey is always visible and allows for multiple votes over the user's entire coding process. For the extended survey, after a user selects an affect they are presented with several categories to optionally annotate their affect with. This data is compiled in a database and extracted to a JSON file for analysis.

3) Data-set Extraction: The compiled data is then processed in Python for analysis. Each user vote is run through a filter that removes invalid votes. Votes that are missing data (frustration score), votes that were cast but missing code snippets, and spam votes (a multitude of votes cast by the same person in a short period of time) are removed.

4) Code Analysis: From the resulting data-set we process the feedback by creating an abstract syntax tree (AST) from code snippets attached to a frustration score. If an AST cannot be parsed from a snippet, the error produced by the compilation of code is caught and added as a feature. The frequencies of AST features, such as Set nodes, Str nodes, and If nodes, are compiled with the frustration score of the feedback. The AST is traversed and each node type counted as a feature. After AST features are extracted, the code snippet is parsed for stylometric and word frequency features. These extracted features are listed in Table I and include feature name, such as AST nodes, along with a description of the feature. The majority of stylometric features we chose to extract are derived from Caliskan-Islam's code features [13].

5) Extended Feedback: For a follow-up study, we extended HappyFace to allow for annotation of the frustration level reported by the learner. We presented the learner with several categories where the learner may feel their frustration can be placed. These categories were created after manually inspecting code reported by frustrated users and hypothesizing the issue faced by each. Also included is an "Other" category that elicits a free-form response from the user. These free-form responses can be used to tune categories presented to the learners once common categories become apparent. A list of these categories, separated into "Happy" and "Frustrated", are listed in Table II.

IV. Online Deployment Studies

We ran two online studies using HappyFace to gather learner feedback and use this feedback to discover frustrating factors in programming. We explored three research questions:

- **RQ1:** Are learners willing to provide feedback about their current emotional state during a learning experience?
- **RQ2:** Can we identify features of code that are related to frustration?
- **RQ3:** Can these features predict categories of problems faced when programming?

A. Surveys

For this paper we ran two implementations of HappyFace to gather frustration data from learners in two studies. The survey used in our first study included the face vote (5-button Happy to Frustrated face choices) and the extraction of the code the participant is working on. After finding several features that correlated with frustration we decided to run a follow-up study to see if these code features can predict the categories of the problems learners were facing. So, for the follow-up study we extended HappyFace to included a second section that allowed the participant to optionally select a problem category (Table II) for the purpose of predicting user chosen categories using the extracted features present in learner code. For our two implementations of HappyFace we integrated our survey with Python Tutor for 2.5 months for the first study and 2 months for our follow-up study. Each survey collected thousands of votes from learners using Python Tutor which were then filtered down to extract usable data.

B. Participants

Any user of Python Tutor could self-report their frustration level and, for the extended survey, annotate their frustration. The Python Tutor user base was a compelling population for discovering frustrating features of code since this user base is likely to be: (1) learners who are new to a language attempting to write small code samples and (2) learners using the visualization features of Python Tutor to better understand the code snippets they are working on. This is because one major source of users on Python Tutor are hundreds of thousands of learners taking introductory programming courses on MOOCs from providers such as Coursera, edX, and Udacity [2]. Since it is unlikely experts with full featured IDEs and experience dealing with common frustrations of code are using Python Tutor, we can attribute frustrations reported through HappyFace to frustrations felt by learners due to features of the code they are working with. Frustrations differ between the coding frameworks being used as a fully featured IDE may have features that eliminate common frustrations of developers, as well as differing between the experience levels of the users being frustrated. A less experienced programmer will find certain features more frustrating than an experienced developer who has discovered solutions to common frustrating problems. Participants voluntarily responded to the HappyFace survey without any prompts or monetary rewards.

C. Analysis

1) Data Cleaning: An initial 7,450 valid votes were received for our first study. We then filtered out any reporting of frustration that was done before the user had started writing code, as we cannot extract code features without this data. After this pruning, we had 1,013 votes with code in Python 2 and 797 votes with code in Python 3. The resultant data contained 83 and 82 different AST node types, 1,025 and 805 word grams, and stylometric features as defined in Table I for Python 2 and 3 respectively. Despite the syntactic rules of both versions of Python being mostly similar, the design of Python 3 changed some features already existing in Python 2 as well as added new features, creating incompatibilities between the two versions. Because of these differences, we needed to separate the analysis of Python 2 and 3 code.

2) System Refinement: After we ran our first study we extend HappyFace's affect survey to take in an annotation by the learner on why they are frustrated. Common reasons for frustration. or non-frustration, are presented to the learner after a face vote. These categories range from the (mis)understanding of code to receiving or fixing of errors. Rather than correlate frustrating features of code, the frustration annotations the learner provides through the extended survey allows us to predict the categories of frustration a learner will feel given a particular set of code features. An initial 909 Python 2 and 1,304 Python 3 votes with code snippets were left after filtering thousands of votes cast by learners for the second study.

V. Deployment Study Results

*1) **RQ1**:* Are learners willing to provide feedback about their current emotional state during a learning experience?

Our first research question is to investigate learners' willingness to inform a system on their current level of frustration while they are in the process of coding. Our goal was to make a survey that could quickly be taken without much interruption to the learner's work flow. If we failed in this goal, learners would be unlikely to give us feedback on their frustration level, so a usable and streamlined survey was presented to learners for their feedback. Further, creating a survey that elicits feedback from the learner removes and necessity for more complex ways to gather frustration (physical reactions, typing characteristics, etc).

During the time period when we ran our first study (Apr 2 – Jun 14, 2016), Python Tutor received 101,044 unique visitors who executed Python 2 or 3 code. Of these visitors, 2,385 voted on the HappyFace affect survey and gave us frustration values and code snippets. This represents a 2.36% participation rate for Python learners on Python Tutor. Our affect survey is meant to be unobtrusive below the coding interface and was completely optional for Python Tutor users to participate, with no reward offered. Despite the affect survey giving users no extrinsic motivation to report their frustration, we received enough data from participants for our analysis of frustrating code features. This showed that learners were willing to give us feedback on their current emotional state despite being deep into the coding process and potentially being frustrated.

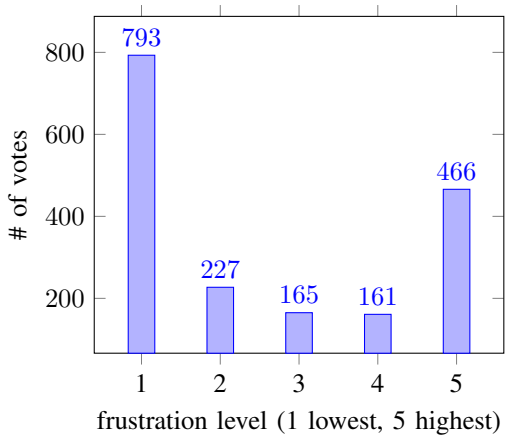

Fig. 3: Frustration level distribution

2.36% of Python users on Python Tutor provided frustration values and code snippets for analysis. This gave us several thousand code snippets to analyze for features related to frustration.

*2) **RQ2**:* Can we identify features of code that are related to frustration?

One of our interests is in the correlation between features of programming languages, in this case Python, with frustration felt by learners. With learner frustration feedback along with features of the code snippets provided through our feedback mechanism we searched for supported features through randomized logistic regression feature selection.

Features were run through randomized logistic regression for discovery of supported features. For Python 2 we found that use of Sets, Globals, Strs, Pow, Slices, and occurrences of SyntaxErrors were selected from the 83 extracted AST features. For Python 3 we found that use of Boolean comparators (is not, !, and !=), import statements, with statements, If statements, Load expressions, and occurrences of SyntaxErrors were selected from the 82 extracted AST features. The strongest correlating feature with frustration was SyntaxErrors. For stylometric features, the average length of the lines of code as well as the usage of Python keywords were found to be supported, with average length having the strongest correlation with frustration. While many wordgrams were selected, the strongest correlations were already detected through AST and stylometric features. The use of for, if, and other Python keywords correlated stronger than any other selected wordgram.

One example of a frustrating feature is the use of global variables. Python has idiosyncratic semantic rules for global variables [20]: If a variable is assigned a value inside of a function, it is considered a local variable unless the variable is declared as global. If the global variable is only referenced inside a function, this declaration is not needed. Not knowing that globals sometimes need to be explicitly declared in Python can cause frustration in learners when it violates their expectation. While the goal of this rule may have been to remove clutter caused by multiple global declarations, it also inserted a frustrating obstacle for learners.

TABLE III: Odds Ratios for Frustrating Code Features

Python 2	
Feature	**Odds Ratio**
SyntaxError	2.28
Str	1.01
Pow	0.59
Slice	1.10
Global	1.60
averageLength	1.02
uniqueKeywords	1.04
Python 3	
Feature	**Odds Ratio**
SyntaxError	2.17
For	1.11
If	1.05
Import	1.75
IsNot	1.78
Not	1.52
NotEq	1.10
averageLength	1.03
uniqueKeywords	1.09
False	1.47
IndexError	2.49
isinstance	2.55

Using randomized logistic regression we found several frustration-inducing features that fell into the following categories: errors, niche language features, and features that increase complexity of the code.

As part of our analysis we extracted the odds ratios of our selected features. The odds ratio "is a measure of association between an exposure and an outcome" [21]. That is, the occurrence of a specific feature will affect the odds of a certain outcome. In the case of HappyFace, the existence of certain AST and stylometric features will increase, decrease, or have no effect on the odds that the learner will be frustrated. If the odds ratio of an independent variable is greater than 1.00, an increase of a unit of the variable will increase the odds that the learner is frustrated. Inversely, when the odds ratio of an independent variable is less than 1.00, an increase in that variable decreases the odds of frustration. The larger the distance from 1.00 (meaning neutral with no effect on odds) the larger effect on the odds of frustration. For example, if the odds ratio of SyntaxError is 2.00, the odds that a learner will be frustrated when a SyntaxError exists in their code is doubled (2.00/1.00). Table III lists the odds ratio of each selected feature for Python 2 and 3 we extracted for the first study. For Python 2, the feature that most increased the odds of frustration was SyntaxError with an odds ratio of 2.28. For Python 3, SyntaxError also had a relatively high odds ratio of 2.17. The highest odds ratio for Python 3 was the use of the isinstance function at 2.55.

An odds ratio analysis supported our findings in RQ2 by confirming that the occurrence of certain features, such as errors, boolean operators, and import statements, increased the odds that the learner is frustrated.

*3) **RQ3**:* Can these features predict categories of problems faced when programming?

Using negotiated agreement [22] on a sample of the first survey's answers to derive hypotheses that could explain why a code snippet was or was not frustrating, we extended our survey component to include annotations for the learner to select as a reason for their current emotional state. In a second experimental survey, we then used these labels to validate our hypotheses. We also included another field so learners could write their own annotations to help us find frustrating experiences we might have not hypothesized. The extension only added a second, but optional, button click to select an annotation, which would not disrupt the learners' willingness to provide feedback.

Since the main purpose of the extended survey was to discover if we could predict the category of the feature causing frustration, the votes of interests are the optional annotations the learners could provide after casting a face vote. For votes with Python 2 code we received 215 annotation votes (23.65% of total votes). For votes with Python 3 code we received 303 annotation votes (23.24% of total votes). The most common reason for being frustrated was **"I get an error"** with 32 occurrences for Python 2 votes and 47 for Python 3 votes. Other commonly chosen reasons were **"My output is wrong"** and **"I don't understand this code"**, both showing that misunderstanding code is a cause for frustration in learners. The most common reason for being happy was **"I understand this code"** with 38 occurrences for Python 2 and 64 for Python 3. Other commonly chosen reasons for not being frustrated are **"My code works"** and **"I fixed an error"**. This was an expected result as the reasons that learners are not frustrated are antithetical to the reasons that learners are frustrated. We also allowed the learner to input their own reason in a free-form text box by selecting the "Other" reason.

Using logistic regression (LR) to predict the learner-selected category based on features extracted from the user's code snippets, we were able to obtain a precision, recall, and f1-scores for predicting categories of frustration selected by learners. LR is a flexible and easy to use method to categorize data [23] that allows us to predict what category of frustration a learner's code likely falls into. Further, we tested other methods of predicting our categories but LR outperformed them all for our dataset, so we selected LR for category prediction. For Python 2 we were able to predict the choice of "I don't understand this code" with a precision of 0.80, recall of 0.33, and f1-score of 0.47 with a support of 12 out of 41 annotations. "I get an error" was predicted with precision of 0.39, recall of 0.85, and f1-score of 0.54 with support of 13 of 41. For Python 3 we were able to predict the choice of "I don't understand this code" with a precision, recall, and f1-score of 0.30 with a support of 10 out of 46 annotations. Selection of "I get an error" was predicted with precision of 0.52, recall of 0.76, and f1-score of 0.62 with support of 17 of 46. The other frustration categories could not be predicted as there was not enough support to do so.

Using logistic regression, we were able to predict when a learner annotated their frustration with "I don't understand this code" with 80% precision.

VI. Limitations

While the ideas underlying HappyFace generalize to any programming language, our current prototype is only for Python. To increase the breadth of our analysis, we would need to parse each language's code into ASTs. Doing so can give us insight on the differences in frustrating features between languages. For example, the metric of numbers of semicolons may affect Java learners much more than Python learners as semicolons are required in Java to terminate lines. Another example is indentation in Python, which affects control flow, so it may correlate with frustration in Python but not in Java.

HappyFace receives data only by learners clicking a button to "vote," but frustration is a continuous experience. Frustration may change as the learner is programming, rising and falling throughout the session. For example, a syntax error may cause little frustration when the learner has just started programming, but their frustration level may increase the longer they have been coding. Since HappyFace collects feedback only at the discrete time of the vote, it cannot characterize affect throughout the entire learning experience. To handle this limitation, it may be beneficial to periodically ping the learner to report their current emotion (i.e., *experience sampling*) so we can see changes over time.

Finally, learners experience frustration differently. For instance, an impatient learner may find even "simple" obstacles more frustrating than their peers do. Also, some votes may represent frustration with external factors; i.e., someone may already be frustrated even *before* coding. Perhaps a learner was having a bad day so encountering a usually-benign obstacle caused them to report they were frustrated. For example, in the follow-up study one learner selected the "Other" annotation and wrote "I have a midterm today" as the reason they were frustrated, which had nothing to do with their code.

VII. Discussion

We discuss implications of our findings from designing and deploying HappyFace and then highlight challenges for future research on detecting learner frustration at scale.

A. Implications of HappyFace Design and Deployment

1) Interventions: We believe HappyFace can be used to inform intervention systems, both automatic and manual. In a large-scale learning framework like a MOOC, a frustrated student can report their frustration. HappyFace can then predict the category of frustration felt by a learner working on the particular code snippet. Interventions to this category of frustration can then be automatically presented to the learner.

For example, when HappyFace predicts that the frustration of the learner is from improper indentation, HappyFace can offer to perform "automatic repair" on the learner's code in order to fix the frustrating code. In the case of our learner Ada from the motivational example (Section II), HappyFace can inform intervention systems on when she is facing a frustrating SyntaxError. The intervention system can then highlight a suggested fix; for Ada's SyntaxError the system can highlight missing parentheses in her print statement. HappyFace can also help in a traditional setting where automatic tools are not available to help the learner. In courses that have human assistants, HappyFace can inform them when a learner is facing frustrations along with the likely features causing frustration. Assistants more suited to helping learners understand code may prioritize helping frustrated learners whose code is difficult to understand. Finally, HappyFace can provide a framework which teachers and platform developers can use to provide interventions that ease learner frustrations. For example, during our second study a few learners stated that they were receiving an indentation error as the reason for their frustration. This feedback could motivate learning platforms to add features such as add auto-indentation or style warnings to assist a learner in properly indenting their Python code.

2) Language Design: Our findings support the need for better feature design in programming languages. Our analysis reinforced the common knowledge that SyntaxErrors are a frustrating obstacle that learners face, and potentially unnecessary in languages targeting novice programmers. Further, code complexity can be monitored and even prevented by the language when a learner's code starts to get too complex, frustrating learners when they do not understand their code. Lastly, certain features and the rules to use them may cause frustration in their implementation, such as global variable usage in Python mentioned in RQ2. HappyFace can discover these frustrating experiences in a programming language so that better designs can be created to remove barriers to programming education and aid learners in their understanding.

3) Feedback Mechanisms: HappyFace shows it is possible to gather meaningful data for discovering insights, such as what features in Python cause learner frustration, by leveraging the actions of the learner while only requiring them to self-report their affect. However, we may want to investigate other ways to collect or encourage feedback. Carter et al. has investigated the automated detection of when a programmer is having difficulty or is "stuck" [24]. Researchers found that it was not efficient to allow programmers to manually change their status to stuck, as developers delay asking for help when they need it. Instead detecting difficulty by monitoring developers can inform helpful interventions earlier. While predicting affect is an important aspect in getting learners help, we are also interested in detecting why learners need help.

4) Prediction of Categories: We were able to predict when a learner did not understand the code because AST feature extraction gives us metrics on how complex the code is, which can create a difficulty in understanding the code for learners. However, while ASTs can be useful for some categories, some AST features cannot give us insight on expected behaviors or values and require more analysis.

Another accuracy-disruptive attribute of the categories is that they can overlap in meaning. Code with an error might also be confusing to the learner, meaning both annotations could possibly be chosen. Once HappyFace discovers these frustration categories caused by confusing code and errors, intervention systems can be informed by HappyFace to produce and present interventions.

Finally, our current extracted features may not be good predictors for the categories, so we must further investigate features that can be extracted from learners' code that could be used to improve category prediction. Despite these current weaknesses, the ability to predict that a learner does not understand the code relates to our finding that complex features cause frustration in learners.

B. Challenges for Future Research

1) Expansion: While Python Tutor is a good initial platform for our prototype, we would like to expand HappyFace's reach into IDEs so that it can collect data on larger-scale programming sessions. The HTML-based survey component of HappyFace allows it to be integrated into any web-based IDE, and we can also port it as a plug-in to desktop IDEs such as Eclipse and Visual Studio. Doing so will let us collect longitudinal data for students working on multiple coding problems over the span of, say, an entire academic term.

2) Extraction of Informative Features: Some of the stylometric and AST features we extracted from learner code correlated with frustration felt by the learner, but it is possible we did not discover and collect features that more strongly correlated with frustration. Further investigation into what frustrates a learner can help discover new features that would be beneficial to extract. One method, implemented in our extended follow-up survey, is to elicit causes of frustration from the learner through free-form text fields. After learners report other causes of frustration, we can extract features we believe correlate with that cause to analyze the strength of correlation the new features have with frustration.

3) Scaling Up Participation: Increasing learner participation in surveys will provide more data on frustrating experiences. There are several possible ways to improve the survey design, including exploring the use of prompts to the learner, or even trying automated detection of frustration. It is possible that knowing automated help could be given by the system could increase participation by learners. We did not want to prompt the learner through a pop-up notification so as to limit the interruption the survey would cause the learner, potentially becoming a frustrating feature itself! In classroom and MOOC settings, learners can be prompted by the instructor to take HappyFace surveys. Despite challenges to increasing participation, the scale of the environments HappyFace can be integrated into, such as millions of learners on MOOCs or Python Tutor, means that even low participation rates can still have thousands of participants providing data.

VIII. Related Work

1) Programmer Frustration: Ford et al. investigated causes of frustration for software developers via a survey [1]. The reported causes of frustration were then grouped into several categories ranging from frustrations caused by the skill of the developer, the complexity of the code, and the features of the programming tools in use. Rather than manual surveys to elicit frustrating features in programming, HappyFace analyzes code with learner-reported frustration to discover frustrating features automatically. Further, HappyFace can predict the problem categories a user is facing based on the feature of the code, removing the necessity of manual categorization of user frustrations previously done and lessening the effort required to discover frustrations felt by programmers.

2) Automated Analysis and Feedback: OverCode is a system that uses static analysis of code to cluster solutions written by learners [25]. Glassman et al. found that OverCode gave teachers a high-level overview that allowed them to discern students' understanding and misconceptions about the code they are working on. OverCode accomplished this by reformatting and "cleaning" (standardizing) raw code, then creating stacks of resulting code that became functionally identical. Teachers or teaching assistants can then observe these stacks for misconceptions and holes in learners' knowledge. The Apex system can automatically provide students explanations for their runtime errors by presenting the learners with the root cause and an explanation for why the cause produces an error [12]. Similarly, Singh et al. presented a method for providing feedback automatically for learner errors by using an error model to present corrections to a learner's code [26]. Both of these methods can be helpful interventions to aid learner understanding of the errors encountered running code.

While these systems provide useful feedback for students, they all require that the programming assignment is known and that a test suite is provided. HappyFace instead analyzes all code, no matter the problem being solved, to discover frustrating features in programming assignments and languages.

3) Crowd-Powered Code Annotations: Codepourri uses a volunteer crowd of learners to create code tutorials and annotate each step of the tutorials [27]. The best annotations for use in a tutorial were chosen through a vote by the learners. These tutorials were judged by experts to be near the quality of their own expert-created tutorials. The importance of this finding to directly inspiring the design of HappyFace is that a crowd of learners can provide a similar quality of observation that experts can and can even provide insights that experts missed. In a similar vein, HappyFace uses a crowd of learners to discover and predict causes of frustration in programming.

IX. Conclusion

Frustrating obstacles during programming caused by lack of instructor support, course difficulty, and even lack of requisite skills can cause learners to drop out prematurely. We presented HappyFace, a lightweight feedback mechanism and automated analysis/prediction engine that can discover these learner frustrations at scale. HappyFace allows an efficient method of data collection from thousands of online users by collecting learner affect and code with a one-button click. We believe that HappyFace's automated analysis and prediction can be used to inform intervention systems built into learning frameworks to provide learners with solutions to frustrations. When these systems lack a specific intervention for a frustration, HappyFace can discover common frustrating features that intervention systems can be extended to solve, which will help scale up feedback in large CS courses and MOOCs.

REFERENCES

[1] D. Ford and C. Parnin, "Exploring the causes of frustration for software developers," *IEEE ICSE 8th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2015.

[2] P. J. Guo, "Online Python Tutor: Embeddable web-based program visualization for CS education," in *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '13. New York, NY, USA: ACM, 2013, pp. 579–584. [Online]. Available: http://doi.acm.org/10.1145/2445196.2445368

[3] D. F. O. Onah, J. Sinclair, and R. Boyatt, "Dropout rates of massive open online courses : behavioural patterns," *EDULEARN14 Proceedings pp. 5825-5834*, July 2014.

[4] G. Conole, "MOOCs as disruptive technologies: Strategies for enhancing the learner experience and quality of moocs, http://www.um.es/ead/red/39/conole.pdf," 2013.

[5] K. Jordan, "MOOC Completion Rates: The Data, http://www.katyjordan.com/moocproject.html."

[6] R. S. J. d. Baker, S. K. D'Mello, M. T. Rodrigo, and A. C. Graesser, "Better to be frustrated than bored: The incidence, persistence, and impact of learners' cognitive-affective states during interactions with three different computer-based learning environments," *Int. J. Hum.-Comput. Stud.*, vol. 68, no. 4, pp. 223–241, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.ijhcs.2009.12.003

[7] A. Repenning, A. Basawapatna, D. Assaf, C. Maiello, and N. Escherle, "Retention of flow: Evaluating a computer science education week activity," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, ser. SIGCSE '16. New York, NY, USA: ACM, 2016, pp. 633–638. [Online]. Available: http://doi.acm.org/10.1145/2839509.2844597

[8] B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer, "What would other programmers do? suggesting solutions to error messages," *CHI 2010*, April 2010.

[9] A. K. Vail, J. B. Wiggins, J. F. Grafsgaard, K. E. Boyer, E. N. Wiebe, and J. C. Lester, "The affective impact of tutor questions: Predicting frustration and engagement," *Proceedings of the 9th International Conference on Educational Data Mining*, 2016.

[10] scikit-learn, "Randomized logistic regression, http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RandomizedLogisticRegression.html."

[11] D. Bieri, R. Reeve, G. Champion, and J. B. Ziegler, "The faces pain scale for the self-assessment of the severity of pain experienced by children: Development, initial validation, and preliminary investigation for ratio scale properties," *Pain*, p. 139150, June 1990.

[12] D. Kim, Y. Kwon, P. Liu, I. L. Kim, D. M. Perry, X. Zhang, , and G. Rodriguez-Rivera, "Apex: automatic programming assignment error explanation," *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2016)*, 2016.

[13] A. Caliskan-Islam, R. Harang, A. Liu, A. Narayanan, C. Voss, F. Yamaguchi, and R. Greenstadt, "De-anonymizing programmers via code stylometry," *SEC'15 Proceedings of the 24th USENIX Conference on Security Symposium*, 2015.

[14] F. Hermans and E. Aivaloglou, "Do code smells hamper novice programming? a controlled experiment on scratch programs," in *IEEE 24th International Conference on Program Comprehension (ICPC)*. IEEE, 2016.

[15] Python Software Foundation, "Python 2 AST library, https://docs.python.org/2/library/ast.html."

[16] ——, "Python 3 AST library, https://docs.python.org/3/library/ast.html."

[17] ——, "Python 2 Builtin library, https://docs.python.org/2/library/__builtin__.html."

[18] ——, "Python 3 builtin library, https://docs.python.org/3/library/builtins.html."

[19] P. Guo, "Python Tutor, http://pythontutor.com/."

[20] Python Software Foundation, "Programming faq, https://docs.python.org/3/faq/programming.html#what-are-the-rules-for-local-and-global-variables-in-python."

[21] M. Szumilas, "Explaining odds ratios," *J Can Acad Child Adolesc Psychiatry*, pp. 227–229, 2010.

[22] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen, "Coding in-depth semistructured interviews problems of unitization and intercoder reliability and agreement," in *Sociological Methods Research*. SAGE, 2013, pp. 294–320. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0049124113500475

[23] M. Pohar, M. Blas, and S. Turk, "Comparison of logistic regression and linear discriminant analysis: a simulation study," *Metodoloski zvezki*, vol. 1, no. 1, p. 143, 2004.

[24] J. Carter and P. Dewan, "Design, implementation, and evaluation of an approach for determining when programmers are having difficulty," *GROUP '10 Proceedings of the 16th ACM international conference on Supporting group work*, 2010.

[25] E. L. Glassman, J. Scott, R. Singh, P. J. Guo, and R. C. Miller, "Overcode: Visualizing variation in student solutions to programming problems at scale," *ACM Trans. Comput.-Hum. Interact.*, vol. 22, no. 2, pp. 7:1–7:35, Mar. 2015. [Online]. Available: http://doi.acm.org/10.1145/2699751

[26] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated feedback generation for introductory programming assignments," *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2013)*, 2013.

[27] M. Gordon and P. J. Guo, "Codepourri: Creating visual coding tutorials using a volunteer crowd of learners," in *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, ser. VL/HCC '15, Oct 2015, pp. 13–21.