Feature Creation to Enhance Explainability and Predictability of ML Models Using XAI

Waseem Ahmed
Faculty of Computing and Information Technology
King Abdulaziz University
Jeddah, Saudi Arabia

***Abstract*—Bringing more transparency to the decision making process in fields deploying ML tools is important in various fields. ML tools need to be designed in such a way that they are more understandable and explainable to end users while bringing trust. The field of XAI, although a mature area of research, is increasingly being seen as a solution to address these missing aspects of ML systems. In this paper, we focus on transparency issues when using ML tools in the decision making process in general, and specifically while recruiting candidates to high-profile positions. In the field of software development, it is important to correctly identify and differentiate highly skilled developers from developers who are adept at only performing regular and mundane programming jobs. If AI is used in the decision process, HR recruiting agents need to justify to their managers why certain candidates were selected and why some were rejected. Online Judges (OJ) are increasingly being used for developer recruitment across various levels attracting thousands of candidates. Automating this decision-making process using ML tools can bring speed while mitigating bias in the selection process. However, the raw and huge dataset available on the OJs need to be well curated and enhanced to make the decision process accurate and explainable. To address this, we built and subsequently enhanced a ML regressor model and the underlying dataset using XAI tools. We evaluated the model to show how XAI can be actively and iteratively used during pre-deployment stage to improve the quality of the dataset and to improve the prediction accuracy of the regression model. We show how these iterative changes helped improve the r2-score of the GradientRegressor model used in our experiments from 0.3507 to 0.9834 (an improvement of 63.27%). We also show how the explainability of LIME and SHAP tools were increased using these steps. A unique contribution of this work is the application of XAI to a very niche area in recruitment, i.e. in the evaluation of performance of users on OJs in software developer recruitment.**

***Keywords*—XAI; ML; AI; Recruitment**

I. Introduction

The use of classification and regression models based on Machine Learning and Deep learning techniques in various domains is now common and ubiquitous. Advances in the field have brought faster diagnosis to patients undergoing life critical treatments in the medical domain, has made autonomous driverless vehicles possible and has revolutionized the commercial and financial world through prediction and recommendation systems. Although these systems are becoming increasingly pervasive in many domains, their adoption has become challenging in areas where trust, accountability and transparency of the decision making process is important and crucial to the users and other stakeholders of the application. One of the main reasons is that most of these ML models are closed *black-boxes* - they lack interpretability and explainability. ML models exist on both side of the spectrum. Models based on linear and logistic regression and classification are simple to understand and decisions made by them are easily explainable; but they lack accuracy and performance. On the other hand, there are ML models built on deep learning technologies which have great accuracy and performance but are not interpretable and decisions made by them are not explainable.

Besides this, there are other inherent problems in these systems. For example, they are trained on existing datasets and tested on an unseen dataset. However, this unseen data is, in reality, not really unseen but available to the developers at development and pre-deployment time. If there is a problem in the data or the data is biased in some way, the model is trained on this incorrect data and decisions it subsequently makes would be biased. And because these systems are trained on a specific data structure, they cannot be generalized or easily extended to systems where structure of data may be different or unknown.

In the early days of AI, systems were knowledge based [1] built on strong formalisms, rules and symbolic representations and thus decisions taken by them could, in most cases, be explained in detail. AI systems today, that are based on subsymbolic representations are skill based [1]. The focus of these systems in largely on performance and lesser on explainability. But for a ML system to be adopted in a life-critical or similar domains the system has to be considered trustworthy and transparent by the domain expert or the end user.

The field of Explainable AI (XAI) has been addressing the issue of bringing more transparency, understandability and explainability to these closed ML models [2], [3], [4], [5]. Bringing trust and explainability to ML systems while eliminating bias in decision making is critical in fields like Human Resources [6].

Systems based on ML are increasingly being used in the business world in decision making, which is sometimes leading to the decision making process being less transparent. Managers may not have the necessary skills to understand the decision making skills of the underlying black-box ML models used to make the decision (prediction or classification) and the XAI system should help them communicate with their team in a natural and more understandable way. XAI also seeks to addresses this concern to ensure that relationships between managers and other decision makers in such domains, which

are based on trust and transparency, are upheld. Ideally, if these systems are deployed in areas like HR, they should be able to bring accountability and address the big issue of eliminating bias during recruitment [7]. Providing a detailed justification when recruiting for a high-profile job may take a legal form and if aided with ML models, and a human understandable explanation will be sought as to why a candidate was accepted or rejected. Making algorithmic decision-making more accountable has been made a part of the Right to Explanation in the European Union General Data Protection Regulation (GDPR) [8].

Recently, HR recruitment in the software industry is increasingly being done using Online Judges through hackathons and online contests [9]. They are increasingly being used by companies to facilitate recruitment at different levels of skills ranging from software developers for various IT projects [10] to freshers in campus recruitment drives. Indeed, there are dedicated OJs primarily used to support the recruitment process like *CodeEval*, *Codility*, *HackerRank*, *HackerEarth* and *Qualified*. Data on these OJs is automatically generated with zero human intervention and is thus devoid of biases related to ethnicity, sex and color, thus, presenting a great opportunity to fields like recruitment where such discrimination is to be avoided.

In this research we show how XAI tools can be better used to bring more understandability and explainability to the recruitment process using OJs. Specifically, we show how XAI tools can be used for two main purposes during pre-deployment:

1) to understand and improve the performance of the ML model.
2) to improve the structure of the underlying dataset such that it contributes to increase the understandability and explainability of the ML model.

We designed and conducted iterative experiments to address both of the above aspects. Our experiences from these experiments are shared in the paper.

The rest of the paper is organized as follows. Section II provides a background on Online Judges (OJs) and XAI in the ML pipeline. Section III describes the first setup of the experiment. Section IV describes how domain knowledge is used to improvise both the model and the dataset. Observations and analysis using XAI on the extended and modified dataset are given in Section V. Related work is given in Section VI followed by the Conclusion section.

II. Related Work

Explainable Artificial Intelligence has drawn a lot of attention recently with lots of survey papers presented in literature on the topic [2], [4], [11], [3] and has drawn a lot of citations. It's importance and significance can be seen by DARPA's increasing interest in the field [12], [13]. Also, there are various domain specific papers that highlight the use and importance of XAI in critical fields like pathology, cancer detection, AI aided Alzheimer's Disease detection and bioinformatics [1], [14], [15], [5], [16]. It has also been used in non life-critical fields like Credit Risk Analysis [17] and in Human Resources in predicting employee attrition and implementing effective employee retention strategies [6]. Although the field of XAI has drawn a lot of attention recently creating new tracks within conferences and workshops and initiating new research initiatives and groups [2], it is not new. Researchers have been working in the field since many decades as detailed in [2].

AI and XAI have recently been used by HR in an increasing number of companies attempting to automate the recruitment process. ML has been used for screening resumes to shortlisting candidates, to analyzing employee churn, implementing strategies to improve employee retention, to identifying training and development needs, and in designing personalized development programs for employees. Some of these approaches have been evaluated by researchers and external agencies for their applicability. For example, [18] has applied the Design Thinking approach to evaluate the explainability aspect of a recruitment system by presenting counterfactuals generated by XAI systems [18]. A group of handpicked volunteers were selected for their experiment. However, their system is pretty basic and considers just two generic personas for evaluation and a relatively small data set.

XAI has also been adopted in fields closely related to software development. For example, in the software development process [19], in defect monitoring and prioritization [20], in software analytics [21] and during software maintenance [22]. There has been ongoing research in predicting developer performance with and without using AI and XAI principles. For example, the work done by [23] closely reflects our work in predicting developer performance without using XAI tools or techniques. Another similarity with our work is that it uses data from Codeforces for analysis. They train four different neural network models and obtain the best results for LSTM with Attention. This work differs from ours in many ways. Firstly, the dataset they use comprises of just 100 users. The paper does not explain how these 100 users were selected from about 140k registered users. The dataset seems too small to be representative of the different categories of users on Codeforces. Secondly, predictability of these users in terms of their behavior between contests is relatively consistent. Also, The paper does not consider temperamental performances of users in contests. Also, in their research, they consider past performances of users in both contests and practice sessions after the contest to predict rating after the contest while we use just contest data.

The work done in this paper differs from other previous works in two main aspects. First, we use XAI to both understand and improve the performance and explainability of the ML model. Second, we apply it to a system to predict developer performance in OJs. To the best of our knowledge, no work exists in this field. Moreover, the research work presented in this paper has a very narrow focus, i.e. evaluation of developer performance prediction during software development which may be in the form of a hackathon session created primarily for recruitment.

III. Background

A. Online Judges

Online Judge (OJ) platforms like SPOJ, EulerProject, Codeforces [24], Codechef, Topcoder, HackerEarth, etc. are being used by a wide spectrum of developers across the world

today to practice, sharpen and showcase their developmental skills. They are being used as development platforms, as crowd-sourcing platforms to hunt for solutions to industrial and science-driven challenges, as online compilers, in education, in workforce development and training, and more recently in recruitment [10] and by coaches in Universities to train and pick the best candidates to participate in the prestigious ICPC Challenge drawing more than 50,000 students from over 3000 Universities worldwide [25].

Most of them regularly hold online rated contests of fixed time durations in which thousands of users across the globe participate. Contestants are expected to solve a certain number of problems in each contest and are awarded *ranks*, *ratings* and *positions* after every contest and these are reflected on the OJ's leaderboard. These are described below in more detail.

1) Problems and verdicts: Each rated contest on an OJs presents contestants with problems at various difficulty levels ranging from elementary (easy) to hard (complex) [26]. In the context of this paper, we denote the difficulty level of a problem with θ ranging from 1 to 5 where θ=1 represents the difficulty level of the easiest problem in the contest and $\theta = 5$ represents the difficulty level of the hardest problem in the contest. Each problem is accompanied by a clear, and unambiguous description statement, sample inputs with their corresponding expected outputs, and time, memory and resource constraints under which the solution to the problem is to be executed to be successfully accepted by the OJ. Simple problems can be implemented with the use of simple logic and simple data structures while hard problems require the use of more sophisticated algorithms and efficient data structures in producing complete and correct solutions in addition to optimization to meet the specified performance and resource constraints.

For each submission made by a user, the OJ returns a verdict which can take one of the values specified in the first column of Table I.

2) Rating and divisions: When users participate in rated contests, they see a positive or negative change in rating that reflects their ability to solve the problems (tasks) in that contest relative to the previous contests. According to the rating, the contestants are split into multiple divisions based on their current rating. Division 1 on Codeforces consists of users who have the highest rankings (1900+), and problems in rated contests at this Division are generally the hardest to solve.

B. XAI Phases in a ML Pipeline

Consider a typical ML Pipeline indicated in Fig. 1 showing the two main phases where the use of XAI tools and techniques can be productive. The first phase is the *Understanding phase*, which subsumes the pre-deployment phases and comprises of Feature Engineering, Model Training and Model Testing. The second phase, is the *Explaining phase*, which subsumes the later two or more post-deployment phases.

The level of detail in the XAI generated explanations vary between the phases. For example, the explanation in the *Understanding* phase can be technical and given in semantics and formalism used by statisticians, mathematicians and informaticians. The first phase involves XAI tools being used

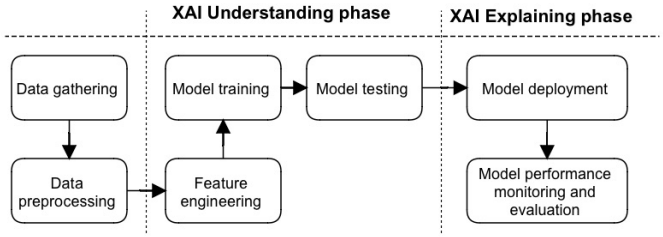

Fig. 1. A typical ML pipeline indicating XAI phases.

by the developer to improve the model during the training and testing stages and ensure that it works as intended [4] when it is deployed in the real-world. In the *Explaining phase*, however, the preferred way of explanation would be in Natural Language and/or intuitive and simple-to-understand visualizations. This phase would involve XAI tools being used by domain experts (a physician or a recruiting agent, for example) and end-users (a patient or a candidate for a job, for example) to interpret and better understand the reasons behind the decisions taken by the ML model deployed in real-life applications [4].

IV. ML Model with Traditional Stats

1) Data collection: For this research, we used *Codeforces* as the choice of OJ as they have easily accessible and large repositories and have a well documented API to access these repositories. First, user profiles of all 143,853 registered users on Codeforces was obtained using the Codeforces API. Next, for each of these users, details of all submissions made by these users to Codeforces was fetched for this study. This included more than 72 million submissions made to Codeforces. Submissions made to rated contests during its specified duration, submissions made to problems for practice, submissions that were successfully accepted or not, and those made in all accepted languages by the judge (including C/C++, Java, Ruby, Rust, Python, PHP, Kotlin, etc.) were included for analysis. The data obtained was largely clean except for missing entries for features that were not relevant to this research.

Features describing each user submission are shown in Table II. To this set of features, two new features - Experience based on time (exp_time) and experience based on the count of rated contests participated by the user (exp_contests) - were calculated and added for each submission entry. *exp_time* was calculated in days as the difference between the current date and the date of registration of the user on Codeforces. *exp_contests* involved more steps. First, to classify contests as rated contest hosted by Codeforces, details of all contests had to be obtained. Next, *contestId* for all Codeforces' rated contests were shortlisted and used to filter user's submissions based on the *problem.contestId* field to only include submissions made to rated contests. Next, using the *author.participantType* field, only submissions made by the user to the contest as a *Participant* were filtered. From this truncated submission list for each user, his *exp_contests* was then obtained.

Next, the nominal variable, *verdict*, in each submission, was converted to a numeric variable $0 \leq score \leq 1$ based on the verdict obtained against the test suite (*testset*) , where $score = 0$ implies that no submission has been made and

TABLE I. OJ RESPONSES TO USER SUBMISSIONS BASED ON QUALITY SUB-CHARACTERISTICS

OJ Verdict	Status of submission's execution	A	B	C
Wrong Answer (*WA*)	wrong output or some requirements not satisfied	X	X	-
Time Limit Exceeded (*TLE*)	exceeded maximal specified processing time limit	-	-	X
Memory Limit Exceeded (*MLE*)	exceeded specified RAM utilization limit (stack or heap)	-	-	X
Runtime Error (*RTE*)	runtime error occurred during execution	-	-	X
Accepted (*AC*)	passed all tests without exceeding resource or time limits	Y	Y	Y

Y - Satisfied, X - Not satisfied A-correct, B-complete, C-meets all specified performance and resource constraints

TABLE II. FEATURES DESCRIBING EACH SUBMISSIONS

#	Feature name	Datatype	#	Feature name	Datatype	#	Feature name	Datatype
1	id	int64	10	timeConsumedMillis	int64	19	author.members	object
2	user	object	11	memoryConsumedBytes	int64	20	author.participantType	object
3	contestId	int64	12	points	float64	21	author.ghost	bool
4	creationTimeSeconds	int64	13	problem.contestId	int64	22	author.startTimeSeconds	float64
5	relativeTimeSeconds	int64	14	problem.index	object	23	problem.points	float64
6	programmingLanguage	object	15	problem.name	object	24	problem.rating	float64
7	verdict	object	16	problem.type	object	25	author.room	float64
8	testset	object	17	problem.tags	object	26	author.teamId	float64
9	passedTestCount	int64	18	author.contestId	int64	27	author.teamName	object

$score = 1$ indicates an *AC*. Concept of *score*, and calculations related to *score* have been elaborated in [27].

A. ML Model Setup

In this experiment, we trained and tested seven ML regression models using traditional developer stats directly provided by OJs and use it to predict the *score* of developer in an unseen contest in a problem (E) at difficulty level $\theta = 5$ which was used as the target variable, Y. Traditional stats used as input features to the ML regression models were as follows: X = [*'rank'*, *'maxRank'*, *'rating'*, *'maxRating'*, *'position'*, *'exp_years*', '*exp_contests*'], The statistical distribution of features in X are given in Table III.

We decided to use symbolic ML representations in this experiments to better assess and study the explainability of a regression model as using subsymbolic models such as Deep Neural Networks or ensembles of simpler regression models will require the deciphering and subsequent understanding of the underlying complex black-box models which still remains a challenge [2].

The seven ML regression models were successfully trained and tested based on data of 143,853 registered users. The RMSE values and the prediction accuracy obtained using the seven ML regression models are given in Table IV. As the purpose of this research was not primarily to improve the prediction of a regression model by tuning the hyperparameters of the model, but to study the effect of features in improving the explainability of a model, we decided to use just one among the seven models. Among the seven ML regression models used, the *GradientBoost* regressor provided the least *RMSE* and the best *score* in our first experiments and was, thus, retained as the model of choice in this research.

The next section explains the evaluation of LIME and SHAP [28], two popular model-agnostic XAI tools that can be easily applied to comprehend ML models based on symbolic representations.

TABLE III. STATISTICS OF TRADITIONAL STATS

Feature	mean	std	min	max
rating	1074.4	454.9	-53.0	3757.0
maxRating	1130.3	473.7	227.0	3979.0
rank	1.0	1.8	0.0	10.0
maxRank	1.2	1.9	0.0	10.0
position	71927.0	41526.9	1.0	143853.0
exp_time	773.2	755.2	4.0	5234.0
exp_contests	12.6	23.7	0.0	816.0

TABLE IV. MODEL PREDICTION USING TRADITIONAL DEVELOPER STATS

ML Model used	RMSE values	accuracy
Elastic	0.1409	0.3106
Gradient Boost	0.1367	0.3507
Lasso	0.1408	0.3118
Linear	0.1407	0.3129
Random Forest	0.1442	0.2780
Ridge	0.1407	0.3129
XGBoost	0.1436	0.2837

B. LIME Analysis

LIME explains a prediction of a machine learning model, in our case the GradientBoost Regressor model, for a query point by finding important predictors and fitting a simple interpretable model. We applied LIME on our model and tested its behavior with ten randomly chosen query points, specifying five as the number of most important predictors to report. Based on these parameters, LIME was applied to our testset; LIME internally generated a synthetic data set, fitted a simple interpretable model of important predictors to it, and then used it to explain the predictions around the specified externally supplied ten selected query points. The explanations generated by LIME for the 10 query points and for our ML regression model are given in Table V.

It can be seen that *position* is consistently being used in all 10 explanations as the feature contributing the most to the prediction. It appears that LIME has identified four crisp categories of positions which are then used in the explanation. The second most important feature appears to be *maxRating* which also appears to have been divided into four separate

TABLE V. LIME EXPLAINABILITY

	Feature contribution 1	Feature contribution 2	Feature contribution 3	Feature contribution 4	Feature contribution 5
1	60871.00 ¡ position ¡= 93407.50	929.00 ¡ maxRating ¡= 1184.00	exp_contests ¿ 17.00	rank ¡= 0.00	870.00 ¡ rating ¡= 1107.00
2	position ¿ 93407.50	maxRating ¡= 929.00	maxRank ¡= 0.00	exp_contests ¡= 2.00	exp_time ¡= 295.00
3	30101.50 ¡ position ¡= 60871.00	1184.00 ¡ maxRating ¡= 1480.50	exp_time ¿ 1139.00	1107.00 ¡ rating ¡= 1422.00	0.00 ¡ maxRank ¡= 3.00
4	30101.50 ¡ position ¡= 60871.00	1184.00 ¡ maxRating ¡= 1480.50	exp_time ¿ 1139.00	0.00 ¡ maxRank ¡= 3.00	0.00 ¡ rank ¡= 3.00
5	position ¡= 30101.50	maxRating ¿ 1480.50	rating ¿ 1422.00	exp_contests ¿ 17.00	exp_time ¿ 1139.00
6	30101.50 ¡ position ¡= 60871.00	exp_time ¿ 1139.00	1184.00 ¡ maxRating ¡= 1480.50	1107.00 ¡ rating ¡= 1422.00	0.00 ¡ rank ¡= 3.00
7	60871.00 ¡ position ¡= 93407.50	929.00 ¡ maxRating ¡= 1184.00	2.00 ¡ exp_contests ¡= 6.00	maxRank ¡= 0.00	870.00 ¡ rating ¡= 1107.00
8	30101.50 ¡ position ¡= 60871.00	1184.00 ¡ maxRating ¡= 1480.50	295.00 ¡ exp_time ¡= 604.00	6.00 ¡ exp_contests ¡= 17.00	1107.00 ¡ rating ¡= 1422.00
9	60871.00 ¡ position ¡= 93407.50	929.00 ¡ maxRating ¡= 1184.00	exp_contests ¿ 17.00	295.00 ¡ exp_time ¡= 604.00	870.00 ¡ rating ¡= 1107.00
10	position ¿ 93407.50	929.00 ¡ maxRating ¡= 1184.00	maxRank ¡= 0.00	rating ¡= 870.00	2.00 ¡ exp_contests ¡= 6.00

categories and are used in the explanation. The exception to this is the 6th datapoint where LIME uses *exp_time* to provide the explanation. For the third and next important features there is a variation between the importance of features in prediction. For example, an XAI output generated in Natural Language to explain the first point may be as follows:

"*Since the user was placed between positions 60871 and 93497.50, and had a rating between 929 and 1184, and had participated in more than 17 contests and ranked at the lowest level, and had a rating between 870 and 1107, ...*"

Similarly, for the second point the explanation would have read as follows:

Since the user was placed at a position higher than 93497.50, and had a rating lower than or equal to 929, and had always been ranked at the lowest level, and participated in no more than 2 rated contests, and made his debut on Codeforces 295 days back , ...

Although this is easily understandable from the domain expert point of view, such explanations involving wide and deep decision trees and which have mathematics at their core, may overwhelm a typical end user even when provided in Natural Language.

To identify the feature importance in explanations, we executed LIME on 3000 separate data points and collated the results as shown in Table VI. Each row in the table shows details of one the six traditional stats, as specified below the table. The columns specify the feature importance. For example, *rating* (Feature 4) is the most important feature used in explaining 2925 of the 3000 points (97.5%) and is used as the second most important feature used to explain 75 of the total 3000 points (2.5%). Another important feature is *maxRating* (Feature 1) which was used as the most important feature to explain 75 of the 3000 data points (2.5%) and the second most important feature in explaining 2558 of the 3000 (85.26%) points. However, we see that the numbers are widely distributed among other features in the table cells. Such distribution of numbers in the table implies that the decision tree used in the explanation by LIME can get very wide clouding the explanation in NL when explaining multiple data points.

A metric that could capture this information could be represented as a k-tuple where k is the number of features used. In this experiment we have used $k = 5$ so we have a 5-tuple as follows: $< 97.5, 85.26, 37.2, 24.97, 19.7 >$ where each number in the tuple indicates the percent contribution of the feature to the k-th position. In this example, we have

TABLE VI. FEATURE IMPORTANCE USING LIME

	1	**2**	**3**	**4**	**5**	%
Feature 4	**2925**	75	0	0	0	20.15
Feature 1	75	**2558**	285	52	26	20.12
Feature 0	0	105	**1116**	762	449	16.33
Feature 5	0	134	650	**749**	591	14.26
Feature 6	0	134	650	749	**591**	14.26
Feature 3	0	4	132	359	695	7.99
Feature 2	0	1	84	296	645	6.89

features = [rating, maxRating, rank, maxRank, position, exp_time, exp_contests]

Feature 4 being used 97.5% of the time as the first feature in explaining a data point and Feature 1 being used 85.26% of the time as the second feature in explaining a data point. In an ideal case (a fully and consistently explainable model), each number in the k-tuple should be 100.

An ideal matrix for XAI would be a left diagonal sparse matrix of size $k \times k$, where k is the parameter to LIME specifying the number of features to include in the explanation.

C. SHAP Analysis

SHAP, like LIME, is model-agnostic and can be applied to explain models based on symbolic representations. It stands for SHapley Additive exPlanations and attempts to explain the prediction of an instance by computing the contribution of each feature to the prediction. We applied SHAP to our regression model and to evaluate its explainability on our dataset. Fig. 2 shows the SHAP plot for the seven traditional stats (listed below the figure). Similar to LIME, SHAP also ranked *position* as its most important feature followed by *maxRating*. But we can see that *exp_contests*, *rating*, and *exp_time* are also highly used.

Having a plot shape which is top-heavy, like an exaggerated overgrown mushroom, helps improve the explainability of an XAI tool. Carefully assembling and curating a good dataset can bring such a shape to the plot and increase its explainability.

V. ADDING DOMAIN KNOWLEDGE TO DATASET

Based on lessons learnt from the previous experiment, we decided to incorporate some domain knowledge directly into the dataset. This first involved basic feature engineering to identify and eliminate unimportant features. Next, having studied different OJ environments, their rated contests and user behavior during contests over the past several years, we decided to include features that implicitly incorporated domain knowledge into the dataset. The next subsections give a few illustrative examples of domain knowledge missing from the

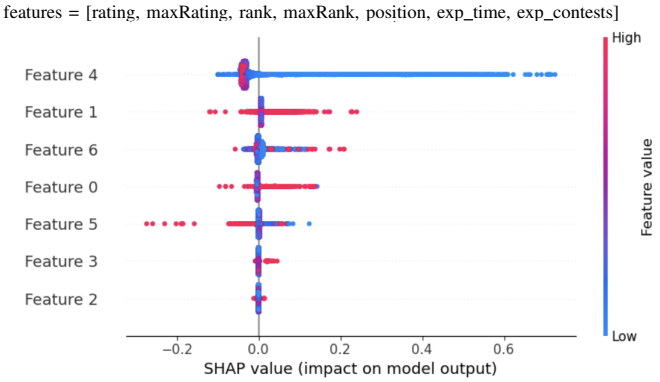

Fig. 2. SHAP Plot showing feature importance for traditional stats.

previous dataset because of which both the predictability of the model and the XAI aspect of the model were affected. This involved introducing new features, deriving new features from existing features, identifying outliers based on these newly generated features, classifying and segregating classes of submissions, we created a new dataset comprising of more than 40 new features in addition to the previous seven traditional stats used in the previous experiment. By including this domain knowledge into the dataset as additional features, and appropriately segregating and curating data, and training the ML regression model on this dataset, we show that the predictability and explainability of the trained model can be substantially increased.

A. Explaining Rank, Rating and Positional Related Inconsistencies

Rank, *rating* and *position* on an OJ may not give a complete picture of the ability and skills of a developer. For example, a low *rank* and *rating* on the OJ does not necessarily imply that the developer has poor programming ability. This is because every new user registering on an OJ, is assigned a default low *rank*, *rating* and *position* regardless of his programming experience or capability. It will take regular participation in rated contests on the OJ and consistently good performances in them for a developer to rise up the leaderboard and to be assigned a high *rank*, *rating* or *position* by the OJ. The rating algorithm used by Codeforces is a variation of the Elo rating [29] used in sports. Understanding the Elo rating system will help explain why an old user on the OJ with good *rating* can fall down the leaderboard with a few bad consecutive performances in contests. This knowledge is not explicitly built into the previous dataset. Identifying and explaining such instances will be challenging to an XAI tool in the existing case. For example, how could an XAI tool explain the situation of a newly registered user on the OJ consistently and successfully solving the difficult problems in contests?

1) Correction mechanism: By associating user submission history to user performances, could help generate better and more reasonable XAI outputs.

B. Upsets *in Contests*

Similar to contests in sports, seeing upsets happen in rated contests is not uncommon. As illustration of upsets, consider the two examples given in Fig. 4. The first example shows the five most recent performances of an average programmer. We can see that he had successfully solved only two easy problems while consistently attempting to solve the third one ($\theta = 3$). However, in the unseen contest he was able to successfully solve the first four problems ($score = 1$) and obtain a high score in problem E (at $\theta = 5$). The second example, shows the performances of a good programmer in the last five most recent contests.We can see that he consistently obtained high scores in all the five problems. But in the unseen contest, however, he performs very badly, not able to successfully solve even the two most easy problems. How could such datapoints be presented by XAI tools?

1) A possible XAI explanation in natural language: "This average user has caused an upset by solving the most difficult problems in the contest."

2) Correction mechanism: These *upsets* or erratic and temperamental performances [27] by contestants can be seen as outliers or stray occurrences in the dataset. Correctly identifying and labelling such *upsets* in the dataset or eliminating them altogether will help improve the accuracy of the model and increase the explainability aspect of the model.

C. Divisions: Different Playing Fields

The *problem.index* field in the *submissions* table (Feature no. 14 in Table II) indicates the problem index (A..E) and hence specifies the problem's difficulty level ($1 \leq \theta \leq 5$). Closer inspection of the dataset reveals a strange trend - higher ranked and rated contestants have difficulty in successfully solving problems D and E, while many lower ranked and rated contestants regularly seem to be successfully solving problem E. Domain knowledge says that all E's are not equal. Users on Codeforces are classified into four divisions based on their rating. Higher rated contestants play (participate) in the highest *Division* (*Division* 1). Rated contests are held separately for each *Division*. While users from higher *Divisions* can register and participate in contests targeted for lower *Divisions*, they are not counted towards their rating. Problems in all rated contests, regardless of *Division*, are indexed from *A* to *E*. This implies that a problem indexed as *E* in a contest at a lower *Division* could be presented as index *A* in a contest at a higher *Division*. This muddles the concept of problem difficulty (θ) and brings inconsistency in the predictability of the regression model. How could such datapoints be presented by XAI tools?

1) A possible XAI explanation in natural language: "This user is from a lower Division and thus could solve problem E which, in reality, may not be too difficult to solve"

2) Correction mechanism: Classifying and segregating submissions based on contestant *rating* into Divisions and possibly training and testing separate ML regression models based on Divisions. This mechanism will ensure the XAI outputs are more natural and understandable.

D. Oscillations Between Divisions

Oscillations between Divisions is an extension of the previous explanation of Divisions. Consider a developer whose current rating is close to the boundary of the next higher Division. A good performance in a rated contest in his existing

(a) Good programmer in good health (b) Good programmer in poor health

(c) Average programmer in good health (d) Average programmer in poor health

(e) A healthy programmer showing consistently good growth (f) A programmer showing a chronic health condition

Fig. 3. Sample health cards showing developer health over a 7-contest window ($w = 7$).

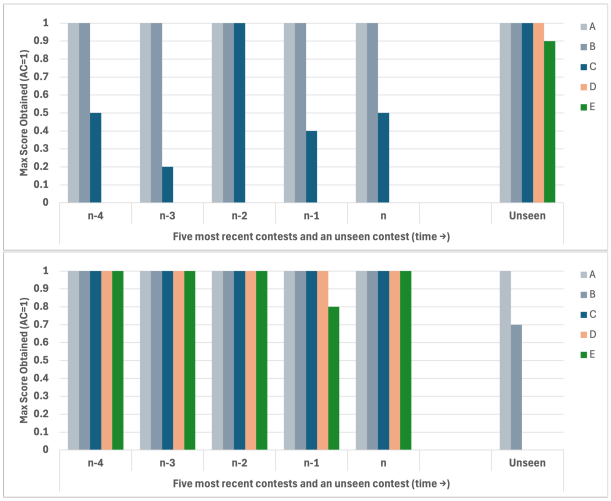

Fig. 4. Upsets in contests.

Division will increase his rating and will subsequently place him in the next higher Division. Similarly, a poor performance by a user in a contest whose rating is close to the boundary of the lower Division, may be placed in the lower Division after the contest. A situation may occur when the ratings fluctuate at the Division boundary causing a user to consistently solve problems D and E when placed in the lower division, and afterward fail to solve problems D and E when placed in the higher Division and vice versa. We refer to this as phenomenon as *oscillations between Divisions*. This common but strange behavior negatively impacts the prediction of a ML model and can be difficult to explain by an XAI tool. How could such datapoints be presented to end user of an XAI tool?

1) A possible XAI explanation in natural language: "*This user's currently rating is at the Division boundary and he is, thus, demonstrating this oscillating behavior.*"

2) Correction mechanism: Specifying a boundary threshold (δ) and segregating users from the dataset whose rating falls within this threshold ($\pm\delta$) at the Division boundary. This mechanism will ensure the XAI outputs are more natural and understandable.

E. Joint Contests

OJs frequently organize joint contests between two different Divisions with problem being identified using the same problem index. However, the rating change calculations differ between divisions. This is not very explicit in the dataset and such contests will have to be identified and marked as such in the dataset.

1) A possible XAI explanation in natural language: "*This user's participation is probably in a joint contest between Divisions and thus has not been able to solve problem E in the higher Division level.*"

2) Correction mechanism: Manually identifying and tagging joint contests as such, will help generate better XAI outputs which are more natural and understandable.

F. Health Card and Development History

Similar to clinical history of a patient [16] and player stats which show past performances and record of a soccer player, a development history could record the history of a developer. This history could provide various insights into a developer's behavior over time. This could be semantically arranged and structured to provide a valuable source of information for XAI applications . Unfortunately, there is currently no standard way of documenting this history - past achievements related to code development, role and contribution to open-source projects, awards obtained in hackathons and programing contests, etc. remain disparate entities and connected together only on the CV of a developer if a job change is desired or to a social media page.

A performance in a single recruitment contest or a hackathon cannot be used as the sole criteria by HR to recruit a candidate. There may be cases of false positive or false negatives.

1) A bad programmer being classified as good - An average candidate may perform extremely well in such a contest if he is lucky enough to get a problem previously 'seen' by him or that he has worked on
2) A good programmer being classified as bad - A good programmer who may not perform well in such a contest because of health-related or personal reasons on that particular day.

However, a record of the developer's recent history augmented with the performance in the recruitment contest can aid HR in better decision making.

1) Reflection of developer health: A health card of a developer provides an clear insight into the health, maturity level and growth of a developer. Fig. 3 shows examples of five health cards. Each health card shows a windows into scores obtained by contestants in their last seven rated contests ($w = 7$) with the rightmost contest being the most recent contest that the user participated in. Each contest has five problems (A..E) at varying difficulty levels (θ) with A being at the lowest level of difficulty ($\theta = 1$) and E at relatively the highest level of difficulty ($\theta = 5$) among the five problems. Good developers are identified by their capability of successfully and consistently solving all five problems with an AC ($score \sim 1$). Average developers are able to solve the first few problems ($\theta \leq 3$) consistently with a $score = 1$, but have difficulty successfully solving problems at the higher level of difficulty ($\theta \geq 4$) and usually end up obtaining $0 \leq score < 0.7$ for these problems.

As can be seen from Fig. 3(a), the first user appears to be good developer as demonstrated by his consistently and (almost) successfully solving all five problems in all his seven past contests. His *rank*, *position* and *rating* on the OJ will continue to remain stable with minor fluctuations. On the other hand, the health card of the second user shown in Fig. 3(b) shows a different trend. The user appears to have a history of solving all five problems in contests. However, his performance in the more recent contests seems to have deteriorated. We refer to this performance deterioration as a sign of *poor health* of the developer. The term *poor health* is not absolute but relative to a developers's past performances. In this example, the user has successfully solved a few problems. But based on his demonstrated capability of solving all problems in the past, we say that his health is *poor*. Because of his poor recent performances, his *rank*, *position* and *rating* will take a beating and fall to levels that may not reflect the developer's actual capability.

Likewise, we have the health cards of two average programmers. Fig. 3(c) shows the health card of a healthy average programmer. We label him *healthy* as his health card shows a consistent performance with the developer consistently solving the first few problems in the most recent seven contests. The *rank*, *position* and *rating* of this developer on the OJ will continue to remain stable with minor fluctuations. On the other hand, the average programmer's health card in Fig. 3(d) shows a deteriorating performance - although he had a history of demonstrated capability in solving problems A and B in a few recent contests, he has consistently failed to successfully solve these problems in the more recent contests. Similar to the effect of deteriorating health of a good developer, the *rank*, *position* and *rating* on this developer on the OJ will drop to levels that do not rightly indicate the capability of this developer.

This *poor health* that we refer to in this paper is similar to the *loss of form* of a player referred to in sports. There is a high probability that a developer in poor health may not be able to perform his best in a recruitment hackathon or a contest much like an *out-of-form* player's performance in a game because of an injury, illness or some personal issue. This developer, as such, should not be evaluated by a single contest in his current poor form.

The health card in Fig. 3(e) shows a developer in good health and demonstrating good and steady growth. We see that his capability as a developer has matured over the last few contests. A few contests back, he was only able to solve problems at $\theta \leq 2$ but has grown in his capability as a developer to successfully solve problems at $\theta \leq 4$ while able to attempt problems at 5.

These health cards in the figure show contests over a window size of $w = 7$. Larger window sizes will give more meaningful and deeper insights into the growth, consistency and maturity of a programmer over a longer time period.

2) Structure of a developer's health card: A health card reflects the performance of a user in each of the rated contests that he has attempted on the OJ. A contest is defined as $C = \{c_1, c_2, ..., c_n\}$ where each c_i gives a summary of the performance of the developer in the contest *i*. The size of the set, $|C|$, gives the contest experience or the number of rated contests the developer has participated in on the OJ. For the scope of this paper, each contest has five problems (A..E) at increasing level of difficulty, with A being at the easiest level ($\theta = 1$) and E at the most difficult level ($\theta = 5$).

Each c in turn consists of sets of 5-tuples represented formally as $c = \{s_1, s_2, ...s_m\}$. where each s is a 5-tuple representing values associated with the five problems (A..E) in the contest. The size of the *set, m*, is contextual and can be varied based on the user need. For example, s_1 could be used to

represent the final verdict awarded by the OJ for the contest where verdicts are assigned representative scores between 0 and 1.0 where 0 indicates that the user has made no submission to the problem in the contest and 1.0 represents an AC with other verdicts (WA, RTE, TLE, MLE, etc.) falling within this range. Similarly, s_2 could be used to record attempts made for each problem in a contest. Other stats that could be stored are numbers of WAs for each problem, number of TLEs obtained for each problem, etc.

For example, consider the following representation of data of a programmer for contest *j*.

$$c_j = \{score_j, attempts_j, wa_j, tle_j, rte_j ...\}$$

where each $c_j \epsilon C$ represents a rated contest that a user has participated in on the OJ. It could contain various details about the performance of the user in problems in the contest at difficulty levels $1 \leq \theta \leq 5$ as 5-tuples. Some of the important statistics that could be included in c_j are given below. *score* is a 5-tuple that gives the maximum score that a user has been able to attain among all attempts that were submitted to the OJ during the contest. Consider the example below:

$$score_j = (1.0, 1.0, 1.0, 1.0, 0)$$

In this contest, the user has been able to successfully solve the first four problems (A..D) but did not solve problem E. The 5-tuple *score* does not indicate the number of attempts (shots) that were taken. The 5-tuple *attempts* indicates the total number of submissions, both successful and unsuccessful, that the user made to the OJ during the contest. Consider the example below

$$attempts_j = (1.0, 1.0, 1.0, 8.0, 0)$$

The user made one submission each to problem A..C, made eight attempts to solve problem D and no attempts at solving problem E. This 5-tuple gives an indication to the ability of a developer to understand the given problem statement. The verdicts returned by the OJ for each of the submissions and for each of the problems are captured in their respective 5-tuples. Consider the 5-tuple for wa_j below

$$wa_j = (1.0, 1.0, 1.0, 1.0, 0)$$

This indicates that the user has made one wrong submission for each of the problems A..D. The 0 for problem could mean two things - either he obtained no WA for problem E for any of the submissions made or he obtained no WA because he made no submission to problem E. This can be better understood from $attempts_j$. Similarly, we have the 5-tuples tle_j and rte_j, which give an indication to the submissions which were awarded TLE and RTE for each of the five problems.

For contestants who have participated in many contests, the size of set C can get large. Also, if we decide to store a large set of data for each contest, the size of the encoding vector can become fairly large. Although this could be seen as a limitation, it adds more explainability to the model. However, this limitation can be overcome by constraining the size of set *C*. The set *C* could be seen in two forms:

1) The set *C* contains the complete history of the developer on the OJ where $|C|$ gives the total sum of rated contests that the developer has participated in. Although this set can get large, this provides insights into the growth and maturity of a developer since his registration on the OJ.
2) The set *C* contains only a window to the most recent *w* contests and provides a glimpse to the health or most recent form of a developer. This can be see in Fig. 3 (Health chart) with where $w = 7$.

[Explain the significance of the two sets] EAI can add transparency and a clearer explanation and justification to the decision making process by presenting the health chart of the developer in a visually intuitive form while additionally providing the set *C*. This will also help in increasing trust in the system.

VI. Enhancing XAI with Derived Data

The main objective of this research was to bring more explainability to the ML models by focusing more on the data aspect rather than just enhancing the performance of ML models though hyper-parameter tuning. This was achieved by enhancing data quality through carefully creating and adding new features to the dataset and by creatively mitigating the effect of noise and outliers using knowledge of the domain.

One of the major changes made to the dataset used in the previous experiment was to incorporate developer history. This proved to be a double-edged sword - it increased the performance of the model while providing more meaningful data to enhance the explainability of the XAI tools. The resultant dataset considerably improved the r2-score of the GradientRegressor model from 0.3507 to 0.9834 (63.27%) which was a substantial improvement.

Additionally, the health chart of a user can be easily extracted from the dataset to add more human-understandable explanation to the XAI generated output.

A. LIME - Results and Discussion

Similar to the previous experiment, we applied LIME to the GradientBoost Regressor model trained on the enhanced dataset and tested its behavior with ten randomly chosen query points, specifying five as the number of most important predictors to report. For these parameters, LIME internally generated a synthetic data set, fitted a simple interpretable model of important predictors to it, and then used it to explain the predictions around the specified externally supplied ten selected query points. The explanations generated by LIME for the 10 query points and for our ML regression model in this experiment are given in Table VII.

Compared to the explanations in the previous experiment, we note that one feature is consistently and dominantly used in all 10 explanations as the feature contributing the most to the prediction and using the same decision. This is reflected in the second and third most important features used in the explanation of the query points with no exceptions. When

generating an explanation for this in Natural Language, the XAI output would be consistent across most, if not all, data points. For example, explanations for the first two data points could read as as follows:

1) "*Since the user could not successfully solve problems B, E and C even though he made feeble attempts to solving problems E and C in the last five contests he participated in .....*"
2) "*Since the user could not successfully solve problems B, E, C and A and made no attempt to solve problem D in the last five contests he participated in.....*"

Compared to the possible explanations that could be generated in the previous experiments, these are more understandable and consistently presented to the end user. A caveat - in the dataset for Codeforces, and generally in most OJs, a large majority of registered participants are not able to successfully solve most problems. This explains the apparent bias visible in the choice of query points in Table VII.

To further identify the feature importance in explanations, we executed LIME on 3000 separate data points and collated the results as shown in Table VIII.

The table has been limited to show just twenty of the total features used in this experiment. Each row in the table shows details of one feature. The columns specify the feature importance. In this experiment, Feature 16 was the most important feature used in explaining all of the 3000 points (100%). The second and thirst most important features were Feature 19 and Feature 17, respectively, which were used to explain all of the 3000 data points (100%). However, we see that the numbers are evenly distributed among other four features in the table cells for the fourth and fifth most important feature. This distribution of numbers in the table differs from the numbers seen in the previous experiment. We can be infer from the features and the range selected that the decision trees used in the explanations may not be very wide which will aid the visual explanation generated by LIME when explaining multiple data points.

In this experiment 5-tuple used to capture the explainability of the model read as follows: $< 100, 100, 99.13, 24.8, 21.8 >$. This is a substantial improvement over the previous 5-tuple which was $< 97.5, 85.26, 37.2, 24.97, 19.7 >$. Comparing the two, we can observe that each feature contribution to the explanation has improved.

Also, and as pointed out earlier, an ideal matrix for XAI would be a left diagonal sparse matrix of size $k \times k$, where k is the parameter to LIME specifying the number of features to include in the explanation. Wee see that this matrix is a left diagonal matrix at $k = 3$ which was not the case in the previous experiment.

B. SHAP: Results and Discussion

We applied SHAP to evaluate the GradientBoost Regressor model trained on the enhanced dataset for explainability. Fig. 5 show the SHAP plot for twenty of the total features present in the dataset. Similar to LIME, SHAP also ranked Feature 16 as the most important feature followed by Feature 19 followed by Features 17, 18 and 15.

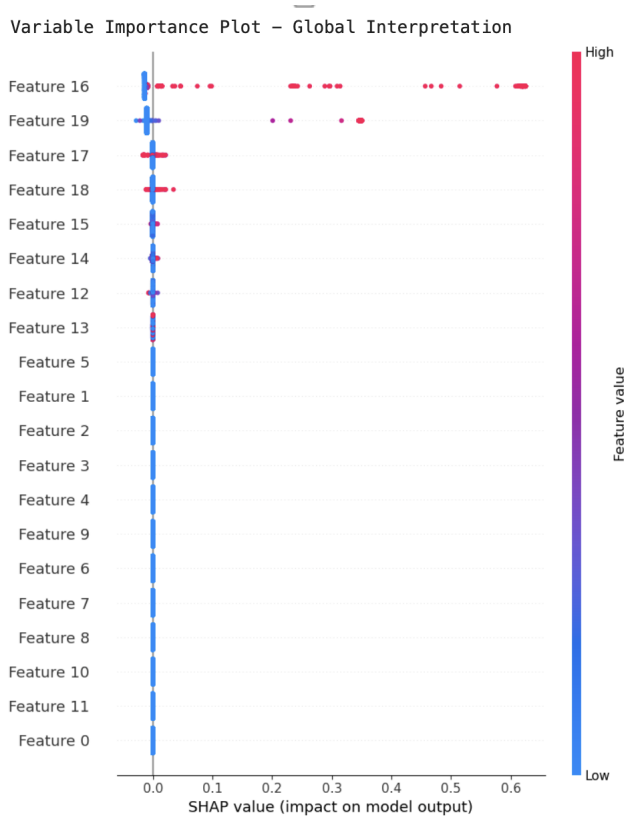

Fig. 5. SHAP values for ML model using the enhanced dataset.

As pointed out earlier, and ideal SHAP plot shape would be a top-heavy one, like an exaggerated overgrown mushroom. The SHAP plot obtained in this experiment appeared to be just that - this greatly aids in improving the explainability aspect of the XAI tool. This was a result of carefully assembling and curating the dataset to bring such a shape to the SHAP plot and, subsequently, to increase its explainability.

VII. Conclusion

Bringing more transparency to the decision making process in fields deploying ML tools is important in various fields. This implies that ML tools need to be designed in such a way that they are more understandable and explainable to the end users while also increasing trust in them. The field of XAI, although a mature area of research, is increasingly being seen as a solution to address these missing aspects of ML systems.

The focus of this work was on improving the transparency of the decision making process in recruitment of software developers using Online Judges. As the field of software development attracts talent at various levels for the high-paying lucrative jobs that is has, it is important to correctly identify and differentiate highly skilled developers from developers who are adept at only performing regular and mundane programming jobs. Also, HR recruiting agents needs to report back to their managers and justify why certain candidates were selected and why some were rejected.

To address this, we built a regressor model that can help differentiate developers based on their ability, while identifying

TABLE VII. LIME EXPLAINABILITY

	Feature contribution 1	Feature contribution 2	Feature contribution 3	Feature contribution 4	Feature contribution 5
1	B_solved_5 ¡= 0.00	E_solved_5 ¡= 0.00	C_solved_5 ¡= 0.00	E_attempts_5 ¿ 1.33	C_attempts_5 ¿ 0.50
2	B_solved_5 ¡= 0.00	E_solved_5 ¡= 0.00	C_solved_5 ¡= 0.00	D_attempts_5 ¡= 0.00	A_solved_5 ¡= 0.00
3	B_solved_5 ¡= 0.00	E_solved_5 ¡= 0.00	C_solved_5 ¡= 0.00	A_solved_5 ¡= 0.00	D_attempts_5 ¡= 0.00
4	B_solved_5 ¡= 0.00	E_solved_5 ¡= 0.00	C_solved_5 ¡= 0.00	A_solved_5 ¡= 0.00	E_attempts_5 ¡= 0.00
5	B_solved_5 ¡= 0.00	E_solved_5 ¡= 0.00	C_solved_5 ¡= 0.00	D_solved_5 ¡= 0.00	E_attempts_5 ¡= 0.00
6	B_solved_5 ¡= 0.00	E_solved_5 ¡= 0.00	C_solved_5 ¡= 0.00	0.50 ¡ E_attempts_5 ¡= 1.33	0.00 ¡ C_attempts_5 ¡= 0.10
7	B_solved_5 ¡= 0.00	E_solved_5 ¡= 0.00	C_solved_5 ¡= 0.00	A_solved_5 ¡= 0.00	E_attempts_5 ¡= 0.00
8	B_solved_5 ¡= 0.00	E_solved_5 ¡= 0.00	C_solved_5 ¡= 0.00	C_attempts_5 ¡= 0.00	E_attempts_5 ¡= 0.00
9	B_solved_5 ¡= 0.00	E_solved_5 ¡= 0.00	C_solved_5 ¡= 0.00	C_attempts_5 ¡= 0.00	E_attempts_5 ¡= 0.00
10	B_solved_5 ¡= 0.00	E_solved_5 ¡= 0.00	C_solved_5 ¡= 0.00	D_solved_5 ¡= 0.00	0.10 ¡ D_attempts_5 ¡= 1.00

TABLE VIII. LIME VALUES FOR DERIVED FEATURES

	1	**2**	**3**	**4**	**5**	**%**
Feature 16	**3000**	0	0	0	0	20
Feature 19	0	**3000**	0	0	0	20
Feature 17	0	0	**2974**	19	2	19.97
Feature 18	0	0	6	**744**	365	7.43
Feature 14	0	0	6	655	**654**	8.77
Feature 12	0	0	5	571	630	8.04
Feature 15	0	0	7	536	643	7.91
Feature 13	0	0	2	475	706	7.89
Feature 0	0	0	0	0	0	0
Feature 1	0	0	0	0	0	0
Feature 2	0	0	0	0	0	0
Feature 3	0	0	0	0	0	0
Feature 4	0	0	0	0	0	0
Feature 5	0	0	0	0	0	0
Feature 6	0	0	0	0	0	0
Feature 7	0	0	0	0	0	0
Feature 8	0	0	0	0	0	0
Feature 9	0	0	0	0	0	0
Feature 10	0	0	0	0	0	0
Feature 11	0	0	0	0	0	0

and ignoring their erratic (temperamental) performances during contests. We showed how both the ML model and the underlying dataset used in training and testing the model can impact the explainability of the model. The underlying dataset that was readily available from the OJ, was enhanced by adding more features and creating new derived features based on our knowledge of the domain. This was done to add more explainability to the model and to increase its predictability and performance accuracy. We also showed how XAI can be actively and iteratively used during pre-deployment to improve the quality of the dataset and to improve the prediction accuracy of the regression model. These iterative changes helped improve the r2-score of our GradientRegressor model from 0.3507 to 0.9834 (63.27%) which was a substantial improvement. We also showed how the consistency and explainability of LIME and SHAP, the XAI tools used in this research, increased over the iterations.

We believe that the work presented in this paper, has great applicability to areas other than developer recruitment. For example, it could be used by project managers to suggest focussed training regimes for developers in their team, to recruit developers for specialized domains, by coaches at Universities to better select their programming team, and in academia where students' performance in programming courses need be predicted to take early remedial action.

REFERENCES

[1] J. Lötsch, D. Kringel, and A. Ultsch, "Explainable artificial intelligence (xai) in biomedicine: Making ai decisions trustworthy for physicians and patients," *BioMedInformatics*, vol. 2, no. 1, pp. 1–17, 2021.

[2] A. Adadi and M. Berrada, "Peeking inside the black-box: a survey on explainable artificial intelligence (xai)," *IEEE access*, vol. 6, pp. 52138–52160, 2018.

[3] A. Das and P. Rad, "Opportunities and challenges in explainable artificial intelligence (xai): A survey," *arXiv preprint arXiv:2006.11371*, 2020.

[4] R. Dwivedi, D. Dave, H. Naik, S. Singhal, R. Omer, P. Patel, B. Qian, Z. Wen, T. Shah, G. Morgan, *et al.*, "Explainable ai (xai): Core ideas, techniques, and solutions," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–33, 2023.

[5] M. Pocevičiūtė, G. Eilertsen, and C. Lundström, "Survey of xai in digital pathology," *Artificial intelligence and machine learning for digital pathology: state-of-the-art and future challenges*, pp. 56–88, 2020.

[6] G. Marín Díaz, J. J. Galán Hernández, and J. L. Galdón Salvador, "Analyzing employee attrition using explainable ai for strategic hr decision-making," *Mathematics*, vol. 11, no. 22, p. 4677, 2023.

[7] D. Hangartner, D. Kopp, and M. Siegenthaler, "Monitoring hiring discrimination through online recruitment platforms," *Nature*, vol. 589, no. 7843, pp. 572–576, 2021.

[8] G. Guizzardi and N. Guarino, "Explanation, semantics, and ontology," *Data & Knowledge Engineering*, vol. 153, p. 102325, 2024.

[9] A. Ru and F. Khosmood, "Hackathons for workforce development: A case study," in *Proceedings of the 5th International Conference on Game Jams, Hackathons and Game Creation Events*, pp. 30–33, 2020.

[10] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, "A survey on online judge systems and their applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–34, 2018.

[11] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, *et al.*, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information fusion*, vol. 58, pp. 82–115, 2020.

[12] G. D. A. DW, "Darpa's explainable artificial intelligence program," *AI Mag*, vol. 40, no. 2, p. 44, 2019.

[13] D. Gunning and D. Aha, "Darpa's explainable artificial intelligence (xai) program," *AI magazine*, vol. 40, no. 2, pp. 44–58, 2019.

[14] V. Vimbi, N. Shaffi, and M. Mahmud, "Interpreting artificial intelligence models: a systematic review on the application of lime and shap in alzheimer's disease detection," *Brain Informatics*, vol. 11, no. 1, p. 10, 2024.

[15] K. Hauser, A. Kurz, S. Haggenmüller, R. C. Maron, C. von Kalle, J. S. Utikal, F. Meier, S. Hobelsberger, F. F. Gellrich, M. Sergon, *et al.*, "Explainable artificial intelligence in skin cancer recognition: A systematic review," *European Journal of Cancer*, vol. 167, pp. 54–69, 2022.

[16] C. Panigutti, A. Perotti, and D. Pedreschi, "Doctor xai: an ontology-based approach to black-box sequential data classification explanations," in *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pp. 629–639, 2020.

[17] A. Gramegna and P. Giudici, "Shap and lime: an evaluation of discriminative power in credit risk," *Frontiers in Artificial Intelligence*, vol. 4, p. 752558, 2021.

[18] H. Sheridan, D. O'Sullivan, and E. Murphy, "Ideating xai: an exploration of user's mental models of an ai-driven recruitment system using a design thinking approach," 2022.

[19] T. Clement, N. Kemmerzell, M. Abdelaal, and M. Amberg, "Xair: A systematic metareview of explainable ai (xai) aligned to the software development process," *Machine Learning and Knowledge Extraction*, vol. 5, no. 1, pp. 78–108, 2023.

[20] Z. Huang, H. Yu, G. Fan, Z. Shao, M. Li, and Y. Liang, "Aligning xai explanations with software developers' expectations: A case study with code smell prioritization," *Expert Systems with Applications*, vol. 238, p. 121640, 2024.

[21] M. A. Awal and C. K. Roy, "Evaluatexai: A framework to evaluate the reliability and consistency of rule-based xai techniques for software analytics tasks," *Journal of Systems and Software*, vol. 217, p. 112159, 2024.

[22] S. Roy, G. Laberge, B. Roy, F. Khomh, A. Nikanjam, and S. Mondal, "Why don't xai techniques agree? characterizing the disagreements between post-hoc explanations of defect predictions," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 444–448, IEEE, 2022.

[23] M. Mahbubur Rahman, B. Chandra Das, A. A. Biswas, and M. Musfique Anwar, "Predicting participants' performance in programming contests using deep learning techniques," in *International Conference on Hybrid Intelligent Systems*, pp. 166–176, Springer, 2022.

[24] Codeforces, "https://codeforces.com."

[25] ICPC, "https://icpc.global."

[26] S. S. Skiena and M. A. Revilla, "Programming challenges: The programming contest training manual," *Acm SIGACT News*, vol. 34, no. 3, pp. 68–74, 2003.

[27] W. Ahmed and A. Harbaoui, "Is this code the best? or can it be further improved? developer stats to the rescue," *IEEE Access*, vol. 12, pp. 144395–144411, 2024.

[28] A. M. Salih, Z. Raisi-Estabragh, I. B. Galazzo, P. Radeva, S. E. Petersen, K. Lekadir, and G. Menegaz, "A perspective on explainable artificial intelligence methods: Shap and lime," *Advanced Intelligent Systems*, p. 2400304, 2024.

[29] Wikipedia, "Elo rating system," September 2024.