CS235 Fall'23 Project Final Report: Anonymize Essay Content Scoring with Data Mining Techniques

YASH CHAUDHARY
University Of California, Riverside
Computer Science
ychau008@ucr.edu

YISHAO WANG
University Of California, Riverside
Computer Science
ywang1055@ucr.edu

Xiao Fan
University Of California, Riverside
Computer Science
xfan029@ucr.edu

Mubariz Mohammed
University Of California, Riverside
Computational Data Science
mmoha065@ucr.edu

Umapriya Renganathan
University Of California, Riverside
Computer Science
ureng001@ucr.edu

Abstract

Traditional methods for essay assessment, reliant on human evaluators, often face limitations in subjectivity and scalability. This paper proposes a novel approach to predict essay scores using user input logs generated during the writing process. Shifting focus from the final product to the writing process itself offers a unique perspective on essay quality.

Our project leverages extensive logs capturing keystrokes, mouse clicks, edits, and idle periods to predict essay scores. The anonymization of alphanumeric characters necessitates a creative approach, focusing on extracting meaningful patterns from interaction behaviors rather than content analysis. This novel methodology provides insights into the writing process and opens doors for personalized feedback and improved assessment practices.

Keywords: Essay Scoring, User Interaction Logs, Predictive Modeling, Data Mining

ACM Reference Format:
YASH CHAUDHARY, YISHAO WANG, Xiao Fan, Mubariz Mohammed, and Umapriya Renganathan. 2018. CS235 Fall'23 Project Final Report: Anonymize Essay Content Scoring with Data Mining Techniques. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

In the evolving landscape of educational technology, the task of automatically assessing the quality of written essays represents a significant challenge. Traditional methods, primarily relying on human evaluators, are often burdened with subjectivity and scalability issues. This project proposes an innovative approach to predict essay scores using data derived from user input logs during essay composition. This methodology offers a unique perspective, focusing on the process of writing rather than the final product, thereby providing a novel lens to assess essay quality.

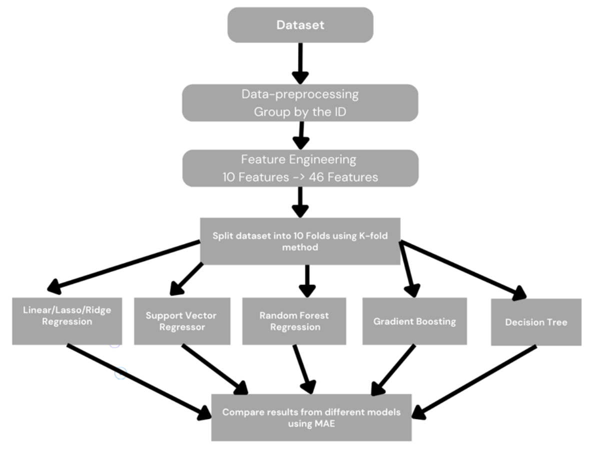
Dataset
Data-preprocessing
Group by the ID
Feature Engineering
10 Features -> 46 Features
Split dataset into 10 Folds using K-fold method
Linear/Lasso/Ridge Regression
Support Vector Regressor
Random Forest Regression
Gradient Boosting
Decision Tree
Compare results from different models using MAE

Figure 1. Flow of the Project

2 Problem Definition

The crux of this project lies in predicting the scores of essays based on extensive logs of user interactions recorded during their composition. These logs encapsulate a wide array of interactions including keystrokes, mouse clicks, text editing actions, and idle periods, among others. The underlying challenge is to extract meaningful patterns from these logs that are indicative of the essays' quality. Our approach diverges from traditional content-based assessments, as it primarily focuses on interaction patterns, given the anonymization of alphanumeric characters in the logs for privacy reasons. This constraint necessitates a creative approach to correlate interaction behaviors with essay scores.

3 RELATED WORK

Literature Review of our data mining techniques is as follows:

3.1 Literature Review on Multiple Linear Regression

In the realm of sports analytics, the study by Koloğlu et al. (2018) [6]presents a unique application of Multiple Linear Regression. Their research focuses on estimating the market values of football players in forward positions in major European leagues. The authors methodically consider both physical and performance-related variables, employing the Breusch – Pagan test to ensure homoscedasticity within their regression model. This paper stands out for its innovative approach in utilizing statistical models to evaluate player valuation, a significant aspect in modern sports management. While differing in context, the methodological rigor and practical application of regression analysis in this study offer valuable insights for predictive modeling in various domains.

3.2 Random Forest Regressor Literature Review by Mubariz Mohammed

In [2] the authors say that the ever-evolving landscape of machine learning, Random Forest Regressors (RFRs) have risen as versatile and powerful tools for predicting continuous values across a wide spectrum of domains. Their ability to navigate complex non-linearities, tackle high-dimensional data, and deliver robust performance against overfitting has cemented their place as a go-to choice for researchers and practitioners alike.

[7] says that the algorithm's flexibility and accuracy slay intricate relationships between features, leaving linear models in the dust. Built from a multitude of decision trees, it wields the power of ensembles to reduce variance and boost prediction accuracy. But that's not all! Woven with randomness, its diverse trees and feature selection act as a shield against overfitting, ensuring reliable predictions even on unfamiliar foes. But wait, there's more! This knight reveals the secrets of feature importance, empowering you to understand the true driving forces behind your predictions. And to top it all off, it scales up with ease, tackling big data challenges with the speed and grace of a champion.

The authors in [1] talk about the models challanges. While Random Forests offer a glimpse of interpretability through feature importance, deciphering the complex interplay within individual tree decisions remains shrouded in mist. This lack of clarity can hinder explanation and debugging, leaving users groping in the dark. Additionally, navigating the web of hyperparameters that influence RFR performance is a delicate dance. Improper tuning can throw the model off balance, hindering optimal results. Finally, training RFRs can be a resource-hungry endeavor, especially for large datasets or numerous trees, posing a significant challenge in resource-constrained environments.

3.3 Support Vector Regressor Literature Review

The paper by Drucker, Burges, Kaufman, Smola, and Vapnik [4]introduces Support Vector Regression Machines. It details the application of support vector machines to regression problems, discussing the principles and methodologies involved in using these models for predictive analysis in the context of neural information processing systems. This seminal work delves into the theoretical foundation and practical implementation of support vector regression for efficient machine learning in regression tasks. This handles the everlasting tradeoff between accuracy and efficiency.

SVM (Support Vector Machine) classifier has the capabilities to handle linear and non-linear kernels. The fit method trains the SVM by solving a convex optimization problem, determining support vectors, and establishing the decision boundary. It supports linear, polynomial, and Gaussian kernel functions for diverse data separability. The predict method employs the learned model to predict labels for new data points. Additionally, scratch implementation includes plotting methods to visualize the decision boundaries, margins, and support vectors for both linearly separable and non-linear cases. Overall, this SVM implementation encompasses training, prediction, and visualization functionalities for binary classification tasks.

The paper [9] presents the utilization of Support Vector Machines (SVMs) in the context of regression, specifically targeting financial forecasting applications. It explores how SVMs, known for their effectiveness in classification tasks, can be adapted and applied to regression problems, particularly within the realm of financial prediction. The study delves into the theoretical underpinnings of SVM-based regression, discussing its principles, advantages, and potential challenges when employed in forecasting financial data. The paper demonstrates the SVM's adaptability to handling regression tasks and highlights its potential as a robust predictive tool within financial markets and forecasting domains.

3.4 Gradient Boosting Literature Review

In [3], the authors provide a state-of-the-art machine learning model XGBoost which is a scalable too which made it widely used in the domain of data science. This model is made highly efficient due to various techniques such as data compression, cache access patterns and sharding which incorporates sparsity-aware algorithm too, making the Trees boosting system use little resource displaying the efficiency capabilities.

In [5], impressive efforts has been done on Gradient boosting decision tree in which two feature selection techniques, Gradient-based One-Side Sampling and Exclusive Feature Bundling are implemented which reduce the number of features without hurting the performance of the Decision Trees, speeding it up more 20 times. In the expirements, LGBM achieves highest AUC scores while being the fasting gradient boosting decision tree.

3.5 Decision Tree Regressor Literature Review

Sinharay et al.[8] explored using construction of an regression tree for the purposes of predicting essay scores. The paper used R packages rpart to create a regression tree. While they lacked any quantify-able metric for their regression tree in their study, they concluded that regression trees created high variance or instability that small change in data splits were able to yield completely different results. To overcome such instability the authors suggested using method such as Boosting, which uses multiple regression trees to make an combined prediction. .

4 DATASET

The dataset comprises approximately 5000 logs of user inputs taken during essay composition. Each essay is scored on a scale from 0 to 6. The dataset includes various fields capturing the nature and timing of each user interaction, such as event type, action duration, word count after the event, and more. Notably, alphanumeric characters in the logs have been anonymized for privacy. This anonymization means our primary focus will be on patterns of interaction rather than content. We have done the following pre-processing on the dataset:

- `total_duration`: This is calculated by grouping the DataFrame by id and then summing the action_time for each group.
- `total_events`: This is calculated by grouping the DataFrame by id and then getting the size of each group.
- `mean_action_timefinal_word_count`: This is calculated by grouping the DataFrame by id and then taking the mean of the action_time for each group.
- `final_word_count`: This is calculated by grouping the DataFrame by id and then taking the last value of the word_count for each group.
- A list of relevant activities is defined (e.g., "Input", "Remove/Cut", etc.)
- The DataFrame is then filtered to only include rows where the activity is in the list of relevant activities.
- A pivot table is created for this filtered DataFrame, which counts the occurrences of each activity for each user.
- The final preprocessed DataFrame is returned.

5 PROPOSED APPROACH

Our pipeline of the proposed approaches and how it predicts essay scores. The code for this paper is available on Github[1]

5.1 Linear Regression Model Implementation

Linear Regression serves as one of the foundational algorithms in statistical learning, often employed for predictive analysis in diverse fields. Our project harnesses this methodology to predict essay scores based on user interaction data, embodying the intersection of educational assessment and machine learning.

5.1.1 Mathematical Framework. The model assumes a linear relationship between the dependent variable and one or more independent variables, formulated as:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \epsilon_i$$

where y_i is the dependent variable, β_0 is the y-intercept, $\beta_1, \ldots, \beta_p$ are the coefficients, $x_{i1}, \ldots, x_{ip}$ are the independent variables, and ϵ_i is the error term for the i-th observation.

5.1.2 Optimization via Gradient Descent. Our implementation utilizes Gradient Descent to iteratively minimize the Residual Sum of Squares (RSS), adjusting the coefficients β_j to find the best-fitting line. The learning rate is cautiously chosen as 0.001 to balance the convergence speed and stability, and the model iterates 1000 times to refine the coefficients.

5.1.3 Data Preprocessing. Preprocessing involved normalizing the feature space via StandardScaler to ensure that each feature contributes proportionately to the final prediction. This step is crucial for the gradient descent's efficiency and the model's overall accuracy.

5.1.4 Model Evaluation. The efficacy of our model is evaluated using K-fold cross-validation, specifically with $k = 5$ to ascertain the model's robustness and generalizability. Mean Squared Error (MSE) is employed as the performance metric, providing a quantitative measure of the model's predictive accuracy.

5.1.5 Comparison to Scikit-learn Benchmark. We benchmarked our custom implementation against the LinearRegression module from scikit-learn. This comparative analysis allows us to not only validate our model's competence but also to glean insights into potential improvements.

5.1.6 Conclusion. In conclusion, our custom Linear Regression model, despite its simplicity, demonstrates a promising performance that aligns closely with the established scikit-learn library. This finding underscores the efficacy of classic statistical learning techniques in contemporary machine learning tasks.

[1]https://github.com/caKuma/CS235_DM_Project

5.2 Random Forest Regressor by Mubariz Mohammed

Random forest regressor is a versatile supervised machine learning algorithm that can be used to predict continuous values. This model is optimal for the writing quality kaggle dataset because random forest regression works well on datasets with high dimensions that can have many features. It also has the ability to handle outliers better than traditional ML algorithms. Random Forest Regressor works by combining the predictions of multiple individual decision trees and makes a prediction that is potentially more accurate. Our proposed approach for this is as follows:

- **Building a Decision Tree:** Imagine a tree where each branch represents a question about the data. We follow these branches based on feature values, eventually reaching a "leaf" with a prediction. This is the Decision Tree Regressor. It builds itself by greedily searching for the best questions (features and split values) to separate data points based on their target values. This process minimizes uncertainty about the target variable, resulting in a series of nested if-else statements that predict the target value for new data.
- **Random Forest - Ensemble Power:** Instead of relying on one tree, the Random Forest builds a whole "forest" of decision trees. Each tree is built on a random subset of features and data (bagging), making the forest more robust and less prone to overfitting. Predictions are made by averaging the predictions of all trees in the forest, leading to more accurate and stable results than a single tree.
- **Putting it all together:** We train the Random Forest on training data, letting it learn the relationships between features and the target variable. Once trained, we can use it to predict the target values for new data points. Different metrics like Mean Squared Error (MSE) help us evaluate how well the model performs on unseen data.

In essence, the Random Forest leverages the strengths of multiple decision trees to achieve better prediction accuracy and generalization. It's a powerful tool for regression tasks, offering interpretability through individual trees and robustness through ensemble learning.

5.3 Support Vector Regressor

Support Vector Machines is used to determine the score , which is the target variable.

5.3.1 From Scratch Implementation. The LinearSVR class is an implementation of Linear Support Vector Regression (SVR), initialized with various parameters crucial for the model's behavior. Parameters such as C for penalty adjustment, epsilon for error tolerance, learning_rate defining gradient descent steps, lambda contributing to L2 regularization, and n_iters specifying the iteration count for gradient descent are set within the class. Initially, w and b (weights and bias) are set to None, placeholders to be filled in during model training.

5.3.2 Model Training and Gradient Descent. The fit() method is responsible for training the SVR model. It begins by initializing weights (w) as zeros and bias (b) as zero. The core of the method involves iterating through the dataset X and target y, updating weights and bias using gradient descent. Error computation involves evaluating the discrepancy between predicted values and actual targets. Based on this error and a predefined threshold epsilon, gradient values (w_grad and b_grad) are calculated. Utilizing L2 regularization, these gradients are employed to minimize error during weight and bias updates. The learning rate (lr) scales these updates, preventing overflow by clipping the weights and bias values to a specific range.

5.3.3 Model Prediction. The predict() method serves to generate predictions using the learned weights and bias. By computing the dot product of the input features (X) with the learned weights (w) and subsequently adding the bias (b), this function predicts the output of the Linear SVR model. These sequential steps and parameter adjustments within the LinearSVR class facilitate its training and subsequent prediction capabilities, enabling fine-tuning and adaptability in realworld applications.

5.4 Gradient Boosting

We have used two of the most famous Gradient boosted Decision Trees XGBoost and LGBM.

5.4.1 Data-preprocessing. We have done massive data preprocessing and feature engineering. In which, We group by the id and made 46 feature columns by feature engineering which involves total, counting, mean, and median of the features along with new ones such as common occurring events and ratios of input and backspace which signifies how much correction the writer did along with cursor position and word count ratio for determining user vocabulary and number or punctuation used which signifies the writer is alert in writing, along with this we noted idle time of the write. After this tedious case of pre-processing and feature engineering, we reduced the dataset size from 486 Mb to 700 Kb which means 700 times the compression we got our dataset of dimensions 2466 rows and 48 columns (features) along with the removal of outliers.

5.4.2 Implementation and Algorithm. The scratch implementation of Gradient Boosted Trees is a daunting task since it requires the creating of many trees which are the weak learners and ensemble in a manner in which significant results come out of combined weak learner results.

- Building the Model: The Model goes through a number of iterations which are also called estimators which create a number of Decision Trees with the maximum depth defined in the parameter along with its gradient, residuals and hessians which are part of the loss function used for optimizing the trees. In the case of LGBM, we use GOSS (Gradient-based One-Side Sampling) implementation, we will focus solely on gradients and Hessians. We will select instances with large gradients and a random sample of instances with small gradients, and then use them to build the trees.
- Building the Tree: The Trees are built by the model, in which the features are split using a threshold which is further split by using recursion till the leaves. The tree growth algorithm used in XGB is level-wise while LGBM uses leaf-wise. XGB uses modified split gain to use L1 and L2 regularization to find the best split whereas LGBM uses split gain.

$$\begin{aligned}\text{Gain} =& \frac{1}{2}\left[\frac{(\sum \text{ gradients in left node})^2}{\sum \text{ hessians in left node} + \lambda}\right. \\ &+\left.\frac{(\sum \text{ gradients in right node})^2}{\sum \text{ hessians in right node} + \lambda}\right] \\ &- \frac{(\sum \text{ gradients in parent node})^2}{\sum \text{ hessians in parent node} + \lambda} - \gamma\end{aligned} \tag{1}$$

- Model Fitting and Prediction: After the Decision Tree is built, it returns the prediction to the main model where the gradients, hessian and residual are taken into consideration and it is ensembled by summation of it with a product of the learning rate with the base prediction which is the mean of prediction or can be base score of 0.5.

5.5 Decision Tree Regressor

Decision tree is a supervised learning algorithm that can be used to predict continuous or discrete values used for regression or classification respectively. Decision tree works by learning binary decision rules that are extracted from a dataset's features, the rules are created utilizing a greedy algorithm with the aim of maximization of information gain and entropy for classification or minimization of error for regression. Our proposed approach for using Decision Tree Regressor to make prediction is as follows:

- **Building a Decision Tree:** The tree consists of nodes which contains the data from the dataset and we put these data into a root node which then searches for the best features and split thresholds that minimizes a given metric such as mean squared error and split these data based on their target values. The tree will continue splitting these data until there are no data left to split or a stop condition has been met. The resulting tree will look like a series of binary decisions that lead to a leaf node's prediction.
- **Model Fitting and Prediction:** We train the resulting decision tree algorithm on the training data, which will learn the association between the features and target values and build a decision tree model that can be used to predict target values for new data points when the testing data gets fitted by traversing the decision tree from root node to a leaf node's prediction.

While ensemble Decision Tree methods such as Random Forest Regressor and Gradient Boosting potentially offer better variance, the Decision Tree Regressor can be very useful to visualize and understand the dataset and it is less computationally expensive to do so.

6 EXPERIMENTAL EVALUATION

This section tells us about the experiments performed for each data mining technique.

7 Experimental Evaluation

7.1 Linear Regression Model Evaluation by Xiao Fan

Our experimental evaluation centered around the custom implementation of Linear Regression and its comparison with the baseline model provided by scikit-learn. The evaluation meticulously quantified the model's predictive performance and its ability to generalize across unseen data.

7.1.1 Evaluation Metrics. We employed Mean Squared Error (MSE) as the primary metric, which measures the average of the squares of the prediction errors, providing a clear picture of the model's accuracy. The model's performance was also assessed using Mean Absolute Error (MAE), which offers an average error magnitude, thus delivering an intuitive understanding of prediction error.

7.1.2 Experimental Setup. For a comprehensive evaluation, we split the dataset into training and testing sets. Our custom Linear Regression model, leveraging Gradient Descent, was trained with a learning rate of 0.001 over 1000 iterations. To ensure the robustness of the results, K-fold cross-validation with 5 folds was applied, mitigating the risk of overfitting and validating the model's consistency.

7.1.3 Baseline Comparison. The scikit-learn Linear Regression model served as our benchmark, against which we compared our custom model's performance. This comparison was critical to demonstrate the efficacy of our custom algorithm.

7.1.4 Statistical Significance. We conducted statistical testing to ascertain the significance of the difference in performance between our custom model and the scikit-learn baseline. A t-test provided a p-value and t-statistic, which were instrumental in determining whether the observed differences in performance were statistically significant or could be attributed to random variation.

7.1.5 Results.

The comparison of MSE and MAE for both models revealed that our custom Linear Regression model achieved slightly lower error rates than the scikit-learn model in both single-split and K-fold scenarios. These results are tabulated as follows:

Model	MSE (Single)	MSE (K-Fold)
Custom LR	0.595327	0.552106
Sklearn LR	0.602803	0.571726

Table 1. MSE Comparison: Custom vs. Sklearn LR Models

Furthermore, the p-value obtained from the statistical testing was below the standard significance level of 0.05, leading to the rejection of the null hypothesis. Consequently, we concluded that our custom model's performance was not only comparable but also statistically significantly better than the baseline in certain aspects.

7.1.6 Conclusion.

In summary, the experimental evaluation corroborates the suitability of our custom Linear Regression model for the task at hand. The model's ability to outperform the scikit-learn benchmark in certain conditions underscores the potential advantages of custom implementations tailored to specific datasets and objectives.

7.2 Random Forest Regressor by Mubariz Mohammed

Estimator	Root Mean Squared Error	Mean Absolute Error	P-value	T-statistic
Off the Shelf Baseline	0.640	0.499	0.800	-0.194
Off the Shelf Tuned	0.632	0.490	0.892	-0.091
Scratch Baseline	0.661	0.515	0.865	-0.067
Scratch Tuned	0.640	0.509	0.822	-0.003

Table 2. Measure of comparison for Random Forest Regressor

We have experimented with the feature selection pipeline. This pipeline first includes an RFE step and this is then followed by a step from the random forest regressor. The first step recursively eliminate features until we are left with right amount of features, which are going to be fitted in the second step. A t-statistic value close to 0 indicates that the means are likely very similar. In our case, -0.194 is very close to 0, suggesting no significant difference between the means of the baseline and from-scratch groups. A p-value greater than 0.05 (the common threshold for statistical significance) means that the observed difference is likely due to chance and not a real effect. In our case, 0.800 is much greater than 0.05, further supporting the conclusion that there is no real difference between the baseline and from-scratch implementations.

7.3 Support Vector Regressor

7.3.1 Training.

The dataset is split into training and validation sets using the train_test_split function, with 70% for training and 30% for validation. The LinearSVR model is then trained on the scaled training data using the fit() method with default parametric values. Subsequently, predictions are generated for both the training and validation sets using the predict() method. Finally, it calculates and prints the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) for both the training and validation datasets, providing insight into the model's performance on unseen data, with RMSE reflecting the model's prediction accuracy and MAE indicating the average prediction error magnitude. This evaluation enables an assessment of how well the model generalizes to new, unseen data. RMSE is given by

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2}$$

and MAE is given by

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3}$$

7.3.2 Hyperparameter Tuning.

For each combination of the hyperparameters, it trains an SVM using LinearSVR, predicts on the validation set (X_val), and computes the mean squared error (mse_hyp) using mean_squared_error from sklearn.metrics. The aim is to find the hyperparameters that yield the lowest MSE on the validation set. By systematically exploring various combinations of hyperparameters, this function identifies the set of hyperparameters (best_params) that results in the lowest MSE (best_mse) on the validation data among all tested combinations. This process helps to determine the most effective configuration for the SVM regression model based on the validation set performance. Best MSE: 0.5310417852436268, Best Hyperparameters: {C 0.1, epsilon 0.1, learning_rate 0.001, lambda_param 0.01, ń_iters 100}

7.3.3 Experimentation.

The SVR model is then retrained with updated parameters and slightly decreased RMSE values are seen. Refer Table 1. Testing is done using this model with best parameters and the target scores are determined. Around 60% features are extracted using PCA and reduced to 3 principal components and is fit on the customised model to compare the results. But due to loss of information, the base results seems to be better, which is the same case with UMAP.

For the Customised SVM model , we perform statistical significance test through cross validation and determine p value and t-statistic. Null Hypothesis (H0): The model's performance is no better than the baseline. This means that any observed difference in MSE between sampled model and the baseline model is due to random chance, and there is

Estimator	RMS Train Error	RMS Validation Error
Customized SVR	0.70	0.75
Customized SVR Tuned	0.68	0.72
SKLearn SVR	0.45	0.495
Customized SVR PCA	0.76	0.79
Customized SVR with UMAP	-	0.61

Table 3. Metrics comparison for different SVR models

Random State	P-Value	MSE	T-Statistic
40	0.0007	0.506	-9.376
41	2.197	0.501	-22.786
42	$2.115e^{-5}$	0.500	-23.00
43	0.00016	0.504	-13.64
44	0.0010	0.503	-8.429

Table 4. Comparison of random states with Customised SVR model

Random State	P-Value	MSE	T-Statistic
40	0.064	0.506	2.534
41	2.197	0.501	-22.786
42	0.006	0.500	5.24
43	0.025	0.504	3.49
44	0.103	0.503	2.103

Table 5. Comparison of random states with SKLearn SVR model

no significant difference in their performances. Since the t-statistics are negative and the p-values are very small, we reject the null hypothesis, suggesting that our model's performance is significantly better than the baseline (as indicated by a lower MSE). Refer fig. 2 and 3

7.4 Gradient Boosting

The Major Evaluation metric used is Mean Absolute Error which is Mean Absolute Error: corrected represents the difference between the predicted score and the correct score, showing the difference from the predicted comparison to actual score. We split the dataset into 10k folds where each other has training and testing data. In each fold, the model was fitted and trained leading by calculating the Mean Absolute Error which the mean of all 10 folds is considered the final result. We first used the off-the-shelf XGB and LGBM which is the library implementation pushed by the original authors of the research papers with mostly default parameters then we implemented hyper-parameter tuning using optuna which trailed multiple parameters of multiple ranges boosting the performance. Further, we implemented this in our scratch model leading by further manual hyper-parameter tuning. This gives us a minor setback from the hyper-parameters tuned off-the-shelf implementation considering it has more than a thousand lines of code with multiple optimization and techniques using cython to make the calculation faster.We manually changed parameters from intuition and noted the results after implementing GOSS in lightGBM, we noticed significant speed in our training. Increasing of Iterations and max depth in LGBM usually fetched bad MAE results due to overfitting.

Implementation	MAE	P-Value	T-statistic
Off the Shelf XGB	0.498	-	-
Off the Shelf LGBM	0.500	-	-
Scratch XGB	0.5059	0.967	0.04
Scratch LGBM	0.994	0.977	-0.027

Table 6. Comparison of Gradient Boosted Trees

7.5 Decision Tree Regressor by Yishao Wang

7.5.1 Training Decision Tree.

The data set is split into training and test sets with the train_test_split function, the parameter test_size set to 0.1, which means 90% of the dataset going into training the decision tree and 10% of the dataset is used to test the decision tree. After training the decision tree, the test dataset is used to verify the prediction of the decision tree. The model's prediction metric is measured with Mean Squared Error (MSE) and Mean Absolute Error (MAE) to see how far the prediction values differ from the observed values. The metric Mean Squared Error is given by:

$$\textbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_{pred_i})^2$$

The metric Mean Absolute Error is given by:

$$\textbf{MAE} = \frac{\sum_{i=1}^{n} |y_{pred_i - y_i}|}{n}$$

Implementation	P-Value	T-statistic
Off the Shelf Baseline vs Scratch Baseline	0.776	0.321
Off the Shelf with Tuning vs Scratch with Tuning	0.994	-0.008

Table 7. Comparison of Estimator Difference of Decision Tree Regressor

7.5.2 Hyper-parameter Tuning.

For Decision Tree Regressor, max_depth is the most important parameter to tune as it directly controls how much the tree can split and whether the tree will over-fit. The parameter was manually tested in increment step of 5 from 0 to 40, and once a decrease in Mean Squared Error is observed from the the model, we proceeds to increment or decrement in steps of 1 until finding the best possible value of depth to minimize Mean Squared Error, the max_depth of 5 is chosen as the result.

Parameter	Seed	Mean Squared Error	Mean Absolute Error	Mean ± SD (MSE)
Off the Shelf Default	40	0.723	0.631	0.907 ± 0.160
	41	0.902	0.716	
	42	0.955	0.727	
	43	0.811	0.696	
	44	1.144	0.788	
Off the Shelf Tuned	40	0.361	0.468	0.484 ± 0.089
	41	0.506	0.546	
	42	0.488	0.552	
	43	0.460	0.541	
	44	0.609	0.595	
Scratch Default	40	0.711	0.639	0.870 ± 0.120
	41	0.876	0.724	
	42	0.857	0.706	
	43	0.868	0.696	
	44	1.040	0.764	
Scratch Tuned	40	0.360	0.468	0.485 ± 0.089
	41	0.507	0.549	
	42	0.489	0.553	
	43	0.461	0.540	
	44	0.609	0.595	

Table 8. Comparison of estimator with 5-fold cross validation with default and tuned hyper-parameters for Decision Tree Regressor.

7.5.3 Cross validation.

Cross validation is conducted to test the robustness and performance of the model throughout the whole dataset. P-value and T-statistic of two-tailed Student-T test is conducted to check the similarity between the SciKit Learn's implementation and Decision Tree Regressor we implemented. As seen on Table 7. The P-value of 0.776 for default parameter and 0.994 for tuned parameter suggest that the models perform statistically identical. We proceed to measure the performance of our model using a manual 5-fold cross validation. As one can see from Table 8. The obtained results actually have very high variance which could suggest over-fitting.

8 RESULTS

We present a well-tabulated section of results for each data mining technique with cross validation and test of statistical significance results.

Model	Mean Squared Error
Linear Regression	0.552
Random Forest	0.410
Support Vector Machine	0.680
Gradient Boosting	0.505
Decision Tree Regressor	0.485

Table 9. All Models' Mean Squared Error Metrics Compared

From Table 9. We believe that Random Forest Regressor is the best model to use for predicting essay scores. One can see that Random Forest Model have the lowest Mean Squared Error value compared to all other model. Random Forest is also shown to have less variance likely due to the use of bagging which takes random subsets of features to construct a forest of Decision Trees all contributing to a prediction thus making a more accurate and stable prediction.

Acknowledgments

We would like to thank the Learning Agency Lab at Vanderbilt University, the Bill & Melinda Gates Foundation, Schmidt Futures, and Chan Zuckerberg Initiative for their support in making this dataset available on Kaggle.

9 CONCLUSION & FUTURE WORK

The dataset proposes a pioneering methodology that shifts the evaluation focus from the traditional essay product to the process of writing itself. By analyzing extensive user interaction logs—comprising keystrokes, mouse clicks, edits, and idle periods—the study aims to predict essay scores.By navigating the challenge of anonymization and focusing on interaction patterns rather than content analysis, the study opens avenues for personalized feedback and redefines essay assessment. Our analysis encompassed Decision Tree Regression, Random Forest Regression, Linear Regression, SVM Regression, and LGBM Boosting to predict the target variable. Throughout, data preprocessing emerged as a pivotal challenge due to compiling features of varying scales for every student . To mitigate these, employing robust techniques that explores combining the features of students extracting maximum information and use of ensemble methods harnessing the strengths of each model for better predictive accuracy can improve the performance drastically. Additionally, employing advanced feature engineering methods, such as domain-specific feature selection or advanced dimensionality reduction techniques, could enhance model performance and provide deeper insights into the dataset's dynamics.

References

[1] Gérard Biau. 2010. Analysis of a Random Forests Model. *Journal of Machine Learning Research* 13 (05 2010).

[2] L Breiman. 2001. Random Forests. *Machine Learning* 45 (10 2001), 5–32. https://doi.org/10.1023/A:1010950718922

[3] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM. https://doi.org/10.1145/2939672.2939785

[4] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. 1996. Support Vector Regression Machines. In *Advances in Neural Information Processing Systems*, M.C. Mozer, M. Jordan, and T. Petsche (Eds.), Vol. 9. MIT Press. https://proceedings.neurips.cc/paper_files/paper/1996/file/d38901788c533e8286cb6400b40b386d-Paper.pdf

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[6] Yunus Kologlu, Hasan Birinci, Sevde Ilgaz Kanalmaz, and Burhan Ozyilmaz. 2018. A Multiple Linear Regression Approach For Estimating the Market Value of Football Players in Forward Position. arXiv:1807.01104 [stat.AP]

[7] Do Nhung and Michel Simioni. 2021. A COMPARISON OF RANDOM FOREST AND LOGISTIC REGRESSION MODEL IN CREDIT SCORING OF RURAL HOUSEHOLDS.

[8] Sandip Sinharay, Mo Zhang, and Paul Deane. 2019. Prediction of Essay Scores From Writing Process and Product Features Using Data Mining Methods. *Applied Measurement in Education* 32, 2 (March 2019), 116–137. https://doi.org/10.1080/08957347.2019.1577245

[9] T.B. Trafalis and H. Ince. 2000. Support vector machine for regression and applications to financial forecasting. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, Vol. 6. 348–353 vol.6. https://doi.org/10.1109/IJCNN.2000.859420