Group 19
Hithisha Dubbaka, Devina Tikkoo, Johnathan Garcia
Github Users: hithishadubbaka, DevinaTikkoo, johnathandanielgarcia

Student Performance Predictor

GitHub Repository: https://github.com/johnathandanielgarcia/Student_Success_Predictor
Video Demonstration: https://www.youtube.com/watch?v=wzqMbxBsu0s

Extended and Refined Proposal

Problem:
We will be building a model that predicts a student's final grade and whether they will pass or fail a course based on school-related data such as study time, past grades, absences, family background, etc. To do so, we will implement logistic regression (LR) and a decision tree (DT) from scratch, comparing the two to determine the optimal algorithm.

Motivation:
Educational institutions often use letter grades and performance on assessments to classify whether students are at risk of underperforming. With an accurate prediction system that accounts for features beyond performance on assessments, institutions could provide tutoring/support for students sooner rather than later. Ideally, students would be assisted before they perform poorly on an assessment. We will be building a simplified version of such a system.

Features implemented:
We will consider the problem solved once we have measured the accuracy, precision, and recall of each algorithm to be floating around ~80%. Accuracy refers to the proportion of correctly classified instances out of the total number of instances. A high accuracy signifies good overall performance; however, it can be misleading with unbalanced data sets, so it is good practice to measure other metrics as well. Precision refers to the proportion of positive classifications that are actually positive; a high precision means that when a model predicts something to be positive, it is likely to be correct. This is important when there is an emphasis on false positives being avoided (i.e. email spam detection). Lastly, recall refers to a models' ability to find all positive classifications; a high recall means that the model is able to identify most positives. This is important when there is an emphasis on false negatives being avoided (i.e. medical diagnosis).

Description of data:
https://www.kaggle.com/datasets/rabieelkharoua/students-performance-dataset/data ~35,000
https://archive.ics.uci.edu/dataset/320/student%2Bperformance? ~19,000

We will have to merge these datasets into one large cohesive dataset, and then randomly generate data that is similar. We will either use a library or AI to help generate the random rows of data that will give us a total of at least 100,000 data points.

Tools/Languages/APIs/Libraries:
We used Python to solve the problem due to its built-in frameworks that allowed smoother data manipulation and visualization. We used **pandas** inside of **Jupyter Notebooks** to preprocess our data, ensuring there were no missing or extreme values that could interfere with model predictions, encoding categorical features as necessary, and performing feature engineering to create points useful for our algorithms. We also ensured our two datasets aligned in their columns. For visualizations, we used frameworks such as **matplotlib**, **seaborn**, and **plotly** to better understand the data and evaluate model performance.

Algorithms Implemented:
We created a Logistic Regression model (LR) to classify students as pass or fail-A, B, and C for pass and D, F for fail- and a Decision Tree (DT) to predict final grade. The LR model uses gradient descent when optimizing the weight vector and minimizes a binary-cross entropy function. It relies on a sigmoid algorithm in order to calculate probabilities to classify in binary. The DT predicts letter grades 0-4, or A-F. The model is also entropy-based and recursively splits the data based on the feature and threshold that will yield the most information gained - greedy-based. Thus, each node evaluates all features and thresholds, and the recursion stops once all samples belong to one class or the maximum depth is reached.

Additional Data Structures/Algorithms used:
Algorithm-wise, we used a recursive approach in several helper functions to fit each model, including gradient descent and entropy-based splitting described above. In main, both models accept active user inputs, which are converted to vector formats, or numpy arrays, to ensure clean, consistent data. Moreover, we utilize the '*train_test_split*' algorithm from scikit-learn to split data into testing/training. We used 80% as a training set and 20% for evaluation as it allowed us to evaluate the performance for both models with "unseen" data.

Distribution of Responsibility and Roles:
All three members collaborated on the Deliverables and source code for Student Success Predictor. Johnathan cleaned, merged, and augmented the data. He also implemented the DT, handling of DT in main, and testing of DT. Devina and Siri collaborated on implementing the LR Model. Devina also implemented the testing of the LR model, while Siri implemented the handling of it in main. For the deliverables, Johnathan recorded the demonstration video and created the ReadME, and Devina and Siri filled out the Report file.

Analysis:

Changes made after Proposal:
In the original proposal, we planned to implement a web-based interface using Streamlit or another platform. However, due to time constraints we were only able to develop a menu-driven Command Line Interface (CLI). Even so, this CLI accepts student data for live predictions while training and testing the models utilizing default or user-defined parameters. Furthermore, we did not originally define how user control would be integrated into the gradient descent logic for Logistic Regression. Subsequently, we allowed users to set the learning rate and max/number of iterations for the model's training in order to provide an interactive experience. Regarding the distribution of roles, we originally had a loose split, but due to conflicting schedules we split roles based on what was needed and whoever was available at that time.

Big O Time Complexity Analysis:
Our implementation included two major algorithm components: Logistic Regression and Decision Tree.

1. **Logistic Regression**

Testing:

- Each iteration of gradient descent processes all n samples with m features.
- Time Complexity: **O(k × n × m)**
    - n = number of training samples
    - m = number of samples
    - k = number of gradient descent iterations

Prediction:

- For each new sample, we calculate a dot product with the weight vector.
- Time Complexity: **O(m)** per prediction

2. **Decision Tree**

Training:

- At each step in the tree, we check all the features and go through all the data points to see where to split the data in a way that best separates the classes.
- In the worst case, if the tree grows in a very uneven way, training can take about **O(n^2 × m)** because the algorithm might have to go through the data many times at different levels of the tree.

Prediction:

- To make a prediction for a new student, the model just follows the branches of the tree until it reaches a final answer at a leaf.
- Time Complexity**: O(h)**, where h is the height of the tree (best case log n, worst case n)

Complexities:

- Logistic Regression: **Training O(k × n × m)**, Prediction **O(m)**
- Decision Tree: **Training O(n² × m)** worst-case, Prediction **O(h)**

Reflection

As a group, working on the Student Performance Predictor was challenging but also impactful. Our goal was to build a model that could help predict whether a student would pass or fail using factors like study time, past grades, and family background. Throughout this process, we were able to combine our individual skills while also learning from each other.

One of the main challenges we faced was handling the data as well and being able to merge the datasets from Kaggle and UCI. This was more complicated because there were differences in format, missing values, and inconsistent feature names. Additionally, we had difficulty with implementing logistic regression and the decision tree algorithm manually rather than using prebuilt libraries. We needed a deeper understanding of the math behind gradient descent, information gain, and debugging. Lastly, pushing to a shared Github repository sometimes led to issues with version control. Overall, all these challenges gave us valuable experience in collaborative coding.

If we were to start the project again we would make changes to the workflow by spending more time in the planning stages and having a clear timeline by using tools like Jira to track tasks. This would help reduce version issues and last minute changes. We would also explore automated data preprocessing, by using Python pipelines, so there wouldn't be delays in model training. Having a clear testing protocol from the beginning would have helped see small errors in the algorithm logic before full integration.

Through this process, each member of the team gained valuable skills and understanding. Devina was able to deepen her understanding of the Logistic Regression model and how its inner algorithms work. Specifically, she learnt how to utilize gradient descent and a Sigmoid function.

In addition she was able to learn how to use SciKit for testing such models and gained a deeper understanding of testing and debugging. Johnathan learned about merging datasets with differences in features, encodings, and size. He also learned about data augmentation and when/how it is used in practice. Further, he learned what a decision tree is, how it serves as a base to more complex ML algorithms (like random forest models), and the inner-workings of it. Hithisha worked on data handling, integrating the logistic regression into the main program, and helped complete the final report. This project strengthened her skills in preprocessing, model integration, and technical documentation.

In conclusion, the Student Performance Predictor was a great learning experience that strengthened our technical skills and teamwork abilities. There were moments of challenge but being able to see our Student Performance Predictor successfully run and compare two different machine learning concepts was rewarding. Our team learned about data handling, algorithm implementation, and collaborative problem solving-skills that we will all use in our future projects.

References

---

https://www.kaggle.com/datasets/rabieelkharoua/students-performance-dataset/data

https://archive.ics.uci.edu/dataset/320/student%2Bperformance

https://en.wikipedia.org/wiki/Decision_tree_learning

https://en.wikipedia.org/wiki/Logistic_regression