University Class Grade Predictor

Sabrina Lawrence

Abstract

For this project, my task was to create a way to predict student grades using their previous grades. The baseline that I used was a weighted average, while the machine learning methods I used were linear regression and matrix completion. My findings were that the linear regression model was the most accurate method for both calculating predicted grades and recommending which class to take, while the weighted average and matrix completion did slightly better than each other in different predictions. The link to the scripts, the dataset, and the commands I used are at the following URL: https://github.com/sabrinakl/university-class-grade-predictor.

1. Introduction

When considering taking a certain class, a student might wonder if they are likely to do well. After all, taking a class that ends up being too difficult or unrelated to past classes can result in the student getting a bad grade, which can hurt the student's GPA and put scholarships or other opportunities at risk. The goal of this work is to create a model that can predict students' performance in a class given their grades in past classes. The input is a spreadsheet of students and their grades, while the output displays each student's predicted grade and real grade for the chosen class and a prediction for which class the student will get the highest grade in. After the implementation of all three methods, I have found that my implementation of linear regression was by far the most accurate for both calculating predicted grades and recommending which class to take. Out of the remaining two models, the weighted average performed better than the matrix completion model in calculating predicted grades, while the matrix completion model performed better in making recommendations.

2. Related Work

I found a paper that achieves something very similar to what I want to do, but they go a lot more in-depth into various approaches (Polyzou & Karypis, 2016). They specifically create two different methods: class-specific regression (CSR), and student-specific regression (SSR). CSR had the drawback of requiring that the same model be applied to all students even though students belonging to different programs may not have taken many of the same classes. SSR was meant to be the solution to this, but its accuracy in grade prediction is limited by the diverse nature of degree plans.

Polyzou and Karypis (2016) also created a matrix factorization model. They seemed to have the same issues with different departments having different structures, so they created a course specific matrix factorization approach by finding subsets of the data that were more dense and populating those first. However, this had varying success based on what department it was applied to. The paper does not state which departments had a higher rate of success, so it is unclear which departments are well suited for a course-specific approach and which ones are not.

The data from my dataset came from one department, so unlike Polyzou and Karypis, I did not run into issues with different levels of accuracy based on the department. The linear regression and matrix completion models I implemented had different focuses, with my linear regression implementation being more student-specific, and my matrix completion algorithm being more class-specific, which is like the different types of models that Polyzou and Karypis used. The students in my dataset all belong to one college's computer science program, so it makes sense that a student-specific approach would perform well for me despite Polyzou and Karypis seeing limitations. As for my more class-specific matrix completion model, the lack of clarity in the paper pertaining to which departments yielded good results made it unclear what to expect when I tried to implement it. After examining the results, it seems that it is not a good fit for this program, and that my student-specific linear regression implementation is more appropriate.

3. Dataset and Evaluation

For my dataset, I am using a collection of grades for various university courses found on Kaggle (Shayan). It contains data for 571 students in over 40 different classes. To train the model, I have randomly split the dataset into train, development, and test sets. The training set consists of 80 percent of the student data (546 students), while the development and tests sets are each about 10 percent of the student data

(57 and 58 students respectively). In order to evaluate the performance of the model, I compared the model's predictions to the real grades that the students received. Because the models used regression and matrix completion, I used RMSE to measure the accuracy of the grade predictions. I also added a second prediction that each model makes, where they also predict which class the student will perform best in. I used a simple measure of accuracy, the number of correct predictions over the total number, in order to evaluate these predictions.

Because not all students take or finish the same classes, there is some missing or otherwise unusable data in the dataset. In the case that the student did not take the class that is being predicted or withdrew or received an incomplete, the missing grade will be imputed. For the baseline, the missing grades are imputed in an extremely simple manner: using the student's CGPA, which is readily available in the dataset. For the regression model, I tried both multivariate feature imputation and K-nearest neighbors imputation. I found that the two yielded similar accuracy, but that K-nearest neighbors ran a lot more quickly, so I ended up using K-nearest neighbors in the model. Since matrix completion models can handle missing data, I used the data as-is instead of performing imputation. If the student withdrew or received an incomplete, it is treated as if the student never took the class, which is filled in along with the other missing grades.

4. Methods

A naive approach without using machine learning is to calculate a weighted average of the student's past grades, with more similar and more difficult classes having a larger impact, and less similar and easier classes having a smaller impact on the prediction. The script I created takes the filename of a CSV file containing the grades data (allowing to to run for train, dev, and tests sets) and the names of the classes for which the predicted grades are desired as command line arguments. The script then calculates the predicted grades for one class at a time. The CSV file is read into a pandas DataFrame, and first specified class column is extracted from the DataFrame. The script then drops that class column from the DataFrame, simulating that the student's grade for the class is unknown. The students' grades, which are in letter grade format, is then translated to a numerical grade using a 4-point GPA scale, which Polyzou and Karypis (2016) also used. The script then iterates over the rows of the DataFrame, and for each row, calculates the weighted average of the student's grades in other classes to predict their grade in the specified class. The weights are calculated by comparing each of the other classes the students have taken to the predicted class. This is done by accounting for both similarity of classes as indicated by the subject and difficulty as indicated by the numerical level. Classes are the same subject if they begin with the same subject code. Classes are similar subjects if they are in the same field. I defined the following fields for the purpose of this baseline that include all of the subject present in the data: Math and Science (Mathematics, Physics, Chemistry, Computer Science), Engineering (Electrical Engineering, Mechanical Engineering, Engineering Fundamentals), Humanities (Humanities and Social Sciences), and Communication (English Language, Technical Communication). Classes are considered unrelated if they are not the same subject nor in the same field. In the weighted average, each term is weighted with x1.5 if the class is in the same field, x1 if the class is in a similar field, and x.5 if the field is completely unrelated to the predicted class. Difficulty is based off of the numerical portion of the class code (100-level class, 200-level class, etc.). Terms of the weighted average are weighed x1.5 if the class is more difficult, x1 if the class is the same difficulty, and x.5 if it is easier than the predicted class. This means that if a class were to be completely unrelated to and easier than the predicted class, it would have a weight of (.5)(.5) = .25 on the prediction. If a class were in the same field and more difficult than the predicted class, it would have a (2)(2) = 4 weight on the prediction. The predicted grade is then translated from a numerical grade back to a letter grade using the 4-point GPA scale. The script outputs the predicted and real grades for each student in the specified class, as well as the final RMSE for the model. It repeats this process for each of the inputted grades.

One machine learning based approach would be to use multivariate linear regression. Because the goal was to predict the grade of a specific student in a given class based on their historical grades in other classes, it is similar to Polyzou and Karypis's student-specific regression model (2016). The model would need to learn the best weights to minimize loss using gradient descent. The model would essentially be learning which classes seem to be related and which ones are not, rather than them being assigned like they are in the baseline. Like in the baseline, the script takes in a CSV file with student grades along with the specified classes. Then for each class, the data is then preprocessed, which involves dropping the predicted class column, one-hot encoding the grades, and imputing missing values with k-nearest neighbors imputation using 5 neighbors and uniform weights. After that, the linear regression model is trained using gradient descent. After the midterm report, I found that I was able to achieve higher accuracy levels using stochastic gradient decent, so I used sklearn's SGDRegressor to help me with this. I used a constant learning rate of 0.005 and maximum iterations is set to 1000. The encoded grades and real grades in numerical form are used as input to the model. The predicted grades are obtained by converting the numerical grades back to letter grades using the same method as the baseline. The script then outputs predicted and real grades for each student, and gives the RMSE between all of the real and predicted grades for that class. Like the baseline, the process is repeated for each inputted class.

Another machine learning based approach would be to use matrix completion, which seems intuitive given that the input is a spreadsheet, with the model filling in the ”missing” grades. Because the implementation is based on the grades of other students in the same class as well as the students' historical grades, it is more-class specific approach. The inputs and data preprocessing is the same as the linear regression model, as well as how the grades are converted from letter to numerical grades. I implemented the matrix completion using collaborative filtering with k=7. My collaborative filtering function decomposes the input matrix using singular value decomposition and then constructs a new matrix with only the most important singular values. The final reconstructed matrix is obtained by multiplying the decomposed matrices with the new matrix. After that, the grades are converted into letter grades and the RMSE between the real and predicted grades is given. This model overcomes the limitations of the previous models in how missing values are not ignored, as they are in the baseline, or separately imputed, as they are in the linear regression model. Since the accuracy of this model is quite a bit lower than linear regression, I would not say that the trade off of not having to impute would be worth it when comparing the use of these two models, but it definitely simplified the implementation of the matrix completion model.

Since collaborative filtering is common for recommendation algorithms, I also wanted to see if the matrix completion model could successfully predict which class the student would do the best in. After the script calculated predicted grades for each class, it also finds which of the grades is the highest to present as a recommendation. After all the predictions have been made, the script calculates the overall accuracy. In order to have something to compare these predictions to, I adjusted my baseline and linear regression scripts also try to predict the classes that each student will perform the best in. and output their predicitons, as well as what percentage of these predicitons are accurate.

5. Experiments

In order to evaluate each model, I will compare their RMSE's in five out of the many available classes: PH-121, CS-105, HS-205, MT-331, and CS-412. This provides a range of subjects and difficulties. I decided to include CS twice because CS classes make up about half of classes taken by the student. While creating the models, it was necessary to adjust the hyperparameters.

First, I looked into the weights I was using for the baseline. I had arbitrarily picked 1.5, 1.0, and 0.5 as weights, but adjusting these slightly could allow for better results. In the following table, I adjusted the weights and looked at the average RMSE across all five classes for both the train and dev sets to see if less extreme (1.25, 1.0, and 0.75), the original (1.5, 1.0, and 0.5), or more extreme (1.75, 1.0, and 0.25) weights would perform better.

Baseline Weight Adjustments		
Weights	Avg. Train	Avg. Dev
1.25, 1.0, and 0.75	0.84	0.73
1.5, 1.0, and 0.5	0.73	0.69
1.75, 1.0, and 0.25	0.85	0.75

Table 1. Shows the results of changing the baseline weights.

After running the model with less extreme, original, and more extreme weights, it is clear the the original weights provide the lowest RMSE on average for both train and dev, meaning that they provide the most accurate results.

I also had to select hyperparameters for the regression model, specifically max iterations and the learning rate. I tried a higher (1500), middle (1000), and lower (500) number for max iterations, as well as a higher (0.1), middle (0.01), and lower (0.005) value for the learning rate. Again, I found the average of the RMSE's of the five courses in order to compare the performances of the hyperparameters.

Regression Maximum Iterations Adjustments		
Maximum Iterations	Avg. Train	Avg. Dev
1500	0.23	0.03
1000	0.21	0.03
500	0.22	0.03

Table 2. Shows the results of changing the regression maximum iterations.

The three values performed exactly the same on dev. Since the maximum iterations value of 1000 performed best on train, I used it for the final model. After choosing a value for maximum iterations, I also chose a value for the learning rate.

Regression Learning Rate Adjustments		
Learning Rate	Avg. Train	Avg. Dev
0.1	1894209118026	97207
0.01	0.21	0.02
0.005	0.25	0.07

Table 3. Shows the results of changing the regression learning rate.

A learning rate of 0.1 was clearly much too large, causing RMSE to skyrocket. The middle learning rate of 0.01 showed also showed better results than the lower learning rate value of 0.005, so I used 0.01 for the final model.

For matrix completion, I had to select the value for the hyperparameter k. In general for collaborative filtering, the system identifies users who have similar preferences and interests, and then recommends items that those similar users have liked or purchased in the past. The number k determines how many of these similar users or items are used to make recommendations. So for the purposes of this model, I needed to see how many similar examples should be used to make calculations for student grades. I tested k values of 5, 6, and 7 to see which would perform the best.

Matrix Completion k Value Adjustments with RMSE		
k	Avg. Train	Avg. Dev
5	0.99	0.79
6	0.99	0.79
7	0.99	0.79

Table 4. Shows the results on RMSE when changing the k value for matrix completion.

The different values of k actually yielded different RMSE's for different classes for each value of k, but the average across the 5 different examples remained the same. Because of this, I decided to compare the percentage of predictions that were correct when running the script for all 5 classes to find which value of k would be best for the final model.

Matrix Completion k Value Adjustments with % Correct		
k	% Train	% Dev
5	61.62%	42.11%
6	61.40%	42.11%
7	61.62%	47.37%

Table 5. Shows the results on percentage of correct recommendation predictions when changing the k value for matrix completion.

A k value of 7 performed the best in dev and was tied with k=5 for the best in train, so I used k=7 for the final model. This means that the 7 most similar examples are used in the calculation of the grades.

After adjusting the hyperparameters, I am able to calculate the final results for the weighted average, regression, and matrix completion models. First, I ran each model's script for all five classes on the train, dev, and test sets so that I could compare the RMSE of each dataset for each course.

Tables 6, 7, and 8 show the RMSE results of the three different models. On average, the linear regression model demonstrated a 96.83% decrease on test set RMSE compared to the baseline, which is a significant improvement. This model likely performed the best because it is the student-specific model. The weighted average model would also be considered student-specific since predictions are based on the students' performance, but since the weights are set and not learned, it is expected that linear regression would be significantly better. On the other hand, the matrix completion model had an average of a 19.79% increase in RMSE compared to the baseline on the test set, so it is actually worse than the baseline for predicting grades. I think a large part of this is due to the model being more class-specific. I think this model overestimates how much the grades of others correlate with the grades of a particular student, which results in a low accuracy, even when compared to a non-machine learning method like the weighted average.

Baseline RMSE's by Course			
Course	Train	Dev	Test
PH-121	0.76	0.85	0.92
CS-105	0.71	0.78	0.77
HS-205	0.74	0.53	0.62
MT-331	0.83	0.78	0.83
CS-412	0.63	0.60	0.65

Table 6. Shows the baseline model's RMSE results of train, dev, and test for each course.

Linear Regression RMSE's by Course			
Course	Train	Dev	Test
PH-121	0.15	0.02	0.02
CS-105	0.16	0.02	0.02
HS-205	0.16	0.02	0.03
MT-331	0.18	0.02	0.03
CS-412	0.12	0.02	0.02

Table 7. Shows the regression model's RMSE results of train, dev, and test for each course.

Matrix Completion RMSE's by Course			
Course	Train	Dev	Test
PH-121	0.91	0.82	0.88
CS-105	1.08	0.87	1.00
HS-205	1.05	0.70	0.84
MT-331	1.02	0.91	1.06
CS-412	0.89	0.65	0.76

Table 8. Shows the matrix completion model's RMSE results of train, dev, and test for each course.

Percentage of Predictions Correct by Model			
Model	Train	Dev	Test
Weighted Average	56.36%	36.84%	50.00%
Linear Regression	88.82%	89.47%	84.48%
Matrix Completion	61.62%	47.37%	55.17%

Table 9. Shows the results percentage of recommendation predictions that were correct for each model.

Table 9 shows the percentage of recommendation predictions that each model got correct for each of the train, dev, and test sets when running the scripts for all five classes. On the test set, the linear regression model had a 68.96% increase in accuracy compared to the baseline, while the matrix completion model had a 10.34% increase compared to the baseline. In this aspect, both the linear regression and matrix completion models were more accurate than the baseline. It makes sense for linear regression to perform well here because its accuracy for calculations was so high. For matrix completion, despite it performing the worse for calculating grade predictions, it did better than the baseline for making recommendations. This makes sense because collaborative filtering is common for recommendation systems, and it is logical that recommendations be made based on the performance of others.

6. Discussion

By looking at the train, dev, and test sets, we can see where certain models perform better and others perform worse. Beginning with the weighted average model, it had 0.73 average RMSE and 56.36% correct recommendations in train, 0.71 average RMSE and 36.84% correct recommendations in dev, and 0.76 average RMSE and 50.00% correct recommendations in test. Based on this, it seems like performance for calculations was similar for each of the datasets, but there is a clear decrease in performance for predicting the class with the highest grade for the dev set. A potential explanation for this is that the dev set had a slightly higher grade average of B+, while the train and test sets had an average of a B grade. This may seem like an insignificant difference, but I think that since all the grades were slightly higher in the dev set, the predicted grades may have been more similar than in the other datasets, resulting in more wrong answers. Despite the overall accuracy of this model not being very good, it performed better than I expected since it is not a machine learning approach.

The linear regression model had 0.15 average RMSE and 88.82% correct recommendations in train, 0.02 average RMSE and 89.47% correct recommendations in dev, and 0.02 average RMSE and 84.48% correct recommendations in test. Despite performing significantly better for calculations on the dev and test sets, the recommendations for train are not much worse than than the recommendations for dev, and the recommendations for test are actually the worst. I think that the difference in RMSE can be explained by how much larger the train set is than the dev and test sets. One potential explanation is that the more data exists, the more the model is forced to generalized as more students with different grades are introduced. Another potential explanation is that more data had to be imputed for the train set since there were more missing values, which may have affected the results. During the midterm report, I was confused that the linear regression model performed so much worse on train than on dev and test, but I think the reasons provided may explain what happened. For grade prediction, the linear regression model performed the worst on test. Test had more missing grades than dev despite being around the same size, so the additional imputation that had to occur may have had an effect. I was pleasantly surprised in how accurate the model is overall, and even more surprised that it is able to do relatively well despite missing grades.

The matrix completion model has 0.99 average RMSE and 61.62% correct recommendations in train, 0.79 average RMSE and 47.37% correct recommendations in dev, and 0.91 RMSE and 55.17% recommendations in test. Despite having having the lowest RMSE and therefore best accuracy for calculating grades on dev, it had the lowest accuracy for making recommendations on dev. The test set, which had higher RMSE and therefore lower accuracy for calculating grades, was actually more accurate than dev for making recommendations. The train set performed the worst in terms of RMSE and the best in terms of correct recommendations. Because train is significantly larger than both dev and test, it seems that this model is not very accurate for calculating grades for larger datasets or making recommendations for smaller datasets. It is better suited for making calculations for smaller datasets or making recommendations on larger datasets. I was initially surprised that accuracy for the calculations and recommendations are not more correlated with each other, but after seeing how the different datasets result in different accuracy values, it makes more sense.

7. Conclusion

Overall, the linear regression model performed very well for calculating grades and recommending classes, performing better than the non-machine learning baseline in both of these aspects. Matrix completion on the other hand, was better suited to making recommendations than calculations, performing worse than the non-machine learning baseline for calculating predicted grades but better for making recommendations. By examining the results, it is clear that a student-specific approach is much better suited for grade predictions within one department. Since matrix completion still performed better than the baseline for making recommendations, I think that it is still good for that problem, but I think that it could have been more accurate if the focus was just on maximizing the accuracy of the recommendations and not on also trying to come up with accurate grade prediction.

Throughout this project, I have been able to deepen my understanding of linear regression, learn new things about data processing and matrix completion, and see for myself how different models can be better suited for different predictions or different datasets. Implementing linear regression on my own in order to solve a problem I proposed is very different than filling in functions or answering questions about it for homework because it required me to not just understand what I was doing, but also why I was doing it. I ended up using a library to help me with the final project, but the process to getting there that I used in the midterm project required a deeper understanding of linear regression than what I had before the project. I am very pleased with the results for my linear regression model, and I think it reflects how well I now understand the concepts behind it. As for data processing, I had never used data frames or one hot encoding before, and now I know how to do these things. Similarly, I had never even heard of matrix completion models before this project, and now I have implemented it on my own. I have also been able to see where it is appropriate to use this model, such as for class recommendations, and when it is not, such as making numerical predictions for grades.

References

Polyzou, A. and Karypis, G. Grade prediction with models specific to students and courses. Technical report, University of Minnesota, Minneapolis, MN, USA, 2016.

Shayan, M. Grades of students. URL `https://www.kaggle.com/datasets/ssshayan/grades-of-students`. Accessed: 2023-03-23.