Analysing Student Data to Determine Factors that Influence Exam Performance

Project Report

submitted towards differentiation for the degree of

Masters of Artificial Intelligence (Msc)

by

Joseph McInerney

Foundations of AI

Word Count: 3220

Queen's University Belfast

February 2025

Contents

1 | Introduction

This project uses Portuguese student data from Cortez and Silva [1] to demonstrate the implementation of mathematical techniques used in artificial intelligence. Cortez and Silva carried out research to try to uncover the factors that caused Portuguese secondary school students to perform so poorly at a European level in both mathematics and Portuguese language. They noted that past exam results were very important for a models' ability to predict the target variable exam results. However, they did also notice that there were significant differences in students performances owing to other factors related to their family life, demographic etc.

It is the goal therefore of this project to reproduce the key findings of this research and demonstrate an aptitude in applying mathematical concepts relating to linear algebra, calculus, probability and statistics to this data set.

The first section of the project is the methodology. Firstly, the information relating to the data set will be picked apart and visualised. Then preprocessing techniques used to prepare the data for analysis will be outlined and reasoning justified. This involves the checking of missing values, encoding categorical variables, standardizing numerical variables and dimension reduction using Singular Value Decomposition (SVD) before attempting to train a model to predict the target variable G3. Afterwords, a model will be trained using the SVD reduced dimension data and the original data to determine whether the dimension reduction successfully reduces computational complexity while maintaining the most important explanatory information.

The Gradient Descent algorithm will then be implemented to accelerate optimal parameter estimation. The importance of the inclusion of previous exam results will be evaluated. Following on from this, a simple Markov Decision Process (MDP) will be defined to model actions that students can take to improve their grades.

2 | Methodology

2.1 Data Collection

The data was downloaded from the UC Irvine Machine Learning Repository [1]. The data was downloaded as a zip file and the arbitrary choice to analyse students results in Portuguese language exams and not mathematics was chosen.

2.2 Exploratory Data Analysis (EDA)

EDA was carried out to provide insightful visualisations related to the data set. The very first step was simply to use the pandas function head() to visualise the first 5 rows of the data frame to get an initial impression of the data.

A correlation analysis was then carried out to infer relationships between variables and their covariance. This was then visualised using the Python library Seaborn's heatmap() function, which displays the strength and direction of correlation between variables with colour. This is very helpful when working with numerous variables, as it enables relationships to be discerned easily from the correlation matrix.

As correlation can only be measured between numeric variables, another method was used to measure the difference in G3 scores achieved by students relative to categorical variables. This involves plotting the distribution of G3 scores in relation to a category. Initially, one can visually analyse the plot to see if there are differences in group means. However, to check for statistical significance, an independent t-test was run using the Python statistical package SciPy. Some features had more than 2 categories, so a function was designed to run pairwise t-tests between all different categories. This test has the following null hypothesis and alternative hypothesis:

H_0 : There is no significant difference of mean G3 score between the two groups of students.
H_a : There is a significant difference of mean G3 score between the two groups of students.

The Analysis of Variance (ANOVA) test was also considered for features with more than 2 categories. This would have indicated whether all the means were equal. It was decided that the independent t-test provides more detail, comparing all groups. This increased number of t-test can be computationally expensive if features have lots of categories. This limitation was judged as not being significant, as the features with the most amount of categories relating to the student's parent's profession had just 5 different categories. Something else to be cautious of is the increase in type 1 errors, where the null hypothesis is falsely rejected as the number of t-tests increases. Therefore, the Bonferroni correction was applied to p values of the t-tests. This involves adjusting α relative to the number of t-tests. The equation below illustrates this, how, as the number of tests increases, α decreases.

$$\alpha_{corrected} = \frac{\alpha}{N}$$

Where, N is the number of t-tests.

2.3 Data Preprocessing

In order to complete Singular Value Decomposition (SVD) all vectors must be on the unit circle to determine the most important dimensions.

2.3.1 Handling Missing Values

The dataset was investigated to determine the presence of missing values. Missing values in data sets can point to errors in data collection or abnormalities in the population that need to be handled. The function isnull() from the python library pandas was used to determine the presence of missing values, and it was found that there were no missing values. As such, no further action needed to be taken. A distribution of the G3 scores was generated using python's matplotlib library as shown in 2.1, and it was noted that there was an abnormality in the data, where there was a disproportionate number of students that scored 0 in the G3 test compared to the otherwise seemingly normal distribution of scores. The observations with scores of 0 were therefore removed to provide the model with more reliable data. These observations were regarded as a form of missing data, as it was not known whether students were really scoring 0 or that the zero denoted their absence for the exam.

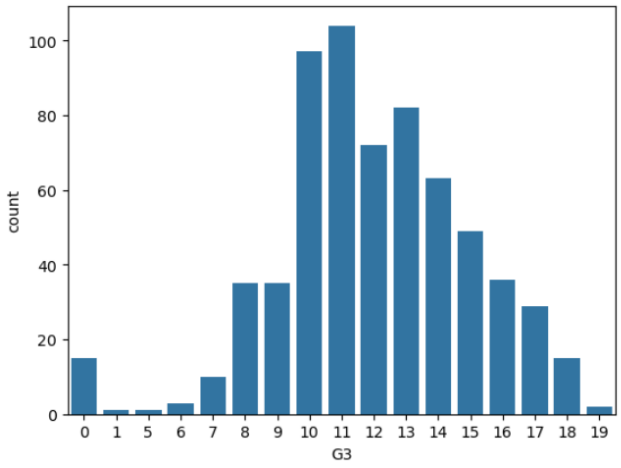

Figure 2.1: A Histogram Displaying a Distribution of G3 Scores

2.3.2 Handling Numerical and Categorical variables

Categorical features were identified and then one-hot encoded. One-hot encoding converts categorical columns to multiple binary columns, where boolean values represent the truth value of a category for the corresponding observation. For this data set, ordinal variables have already been encoded numerically like study time ordered from 1 to 4 where 1 is the least amount of study time and 4 is the most. This prevents the loss of the notion of order if one-hot encoded. One-hot encoding also increases the number of dimensions of the data set, but this will be curtailed with Principal Component Analysis (PCA). Another issue with one-hot encoding is the production of a more sparse matrix, but this should not pose a problem with the chosen methodology.

Numerical features are standardized using the scikit-learn python library, which leverages the following equation to standardize values using the mean (μ) and standard deviation (δ).

$$z = \frac{x - \mu}{\delta}$$

This transformation of features by standardisation and one-hot encoding can be implemented effectively and clearly using scikit-learn's Column transformer, as shown in the code snippet below.

```
col_trans = ColumnTransformer(
    [
        ('one_hot', OneHotEncoder(), make_column_selector(dtype_include=object)),
        ('standardize', StandardScaler(), make_column_selector(dtype_include=np.
            number))
    ]
)
X_scaled_and_encoded = col_trans.fit_transform(X)
```

Listing 2.1: Column Transformation in Python

2.4 Feature Reduction using SVD

The goal of feature reduction is to reduce the number of dimensions of the independent variables to reduce time complexity while maintaining as much information as possible responsible for the variance of the dependent variable. To achieve this, a matrix M is factorized into three constituent parts using SVD denoted by the following equation:

$$M = U\Sigma V^T$$

Where U is an orthogonal matrix containing the left singular vectors, Σ is a diagonal matrix containing the singular values, and V^T is the transpose of the matrix containing the right singular vectors.

After performing SVD on the standardized and encoded matrix, the explained variance ratio was calculated, which indicates how much variance each singular value captures in the original data. The explained variance for each principal component i is computed as follows:

$$\text{Explained Variance}_i = \frac{s_i^2}{\sum_{j=1}^{k} s_j^2}$$

Where s_i represents the singular values obtained from the decomposition. A plot of the explained variance ratio per component is generated to visualize how much variance each component captures.

A threshold of 95% cumulative variance was set, which means the minimum number of components k that explain at least 95% of the total variance will be retained. The number of components k is determined by finding the point at which the cumulative sum of the explained variance exceeds the 95% threshold:

$$k = \arg\max\left(\text{Cumulative Explained Variance} \geq 0.95\right) + 1$$

The right singular vectors corresponding to the top k singular values are retained, and the data is projected into this reduced-dimensional space. This results in a transformed dataset, where the number of features has been reduced while maintaining the majority of the information relevant for variance in the dependent variable.

Finally, the original data matrix is reconstructed into the reduced-dimensional space by multiplying it with the reduced matrix of right singular vectors:

$$X_{\text{SVD}} = X \cdot V_h^{T_{\text{reduced}}}$$

Where $V_h^{T_{\text{reduced}}}$ contains only the top k right singular vectors. This transformation yields a dataset with fewer features, optimized for model training, while preserving the most significant information.

2.5 Parameter Optimisation

Gradient descent was implemented manually to show understanding of the mathematical concepts it includes. Gradient descent computes the derivative of the loss function to determine in which direction to tweak the parameters to minimise loss (error). It was used in this project to optimise the parameters in for linear regression in predicting G3 scores, in line with the methodology outlined by Cortez and Silva, who also used regression for the estimation of this target variable. Essentially, the parameters of the linear regression model find the line of best fit that minimises errors in the model,

2.5.1 Code Implementation of the Mathematical Concept

The following key lines of code implement gradient descent for linear regression, closely following the mathematical principles of optimization.

```
predictions = X.dot(theta)
```

This line calculates the predicted values of the target variable, $\hat{y}$, for the given input features X, corresponding to:

$$\hat{y} = X \cdot \theta$$

where X is the matrix of input features, and θ is the vector of model parameters. The dot product represents the sum of feature values weighted by the respective parameters for each observation.

```
errors = predictions - y
```

Here, the error terms are calculated as the difference between the predicted values $\hat{y}$ and the actual target values y, which gives us:

$$\text{errors} = \hat{y} - y$$

This reflects the difference between the model's predictions and the observed data, which gradient descent seeks to minimize.

```
gradients = (1/m) * X.T.dot(errors)
```

The gradient of the cost function with respect to θ is computed here. The formula for the gradient is:

$$\text{gradients} = \frac{1}{m} X^T \cdot (\hat{y} - y)$$

where m is the number of training examples, and X^T is the transpose of the input feature matrix. This gradient points in the direction of the steepest ascent, so to minimize the cost function, the parameters are updated in the opposite direction of the gradient.

```
theta -= learning_rate * gradients
```

This line performs the parameter update step. It adjusts the parameters θ by moving them in the direction opposite to the gradient, scaled by the learning rate α:

$$\theta = \theta - \alpha \cdot \text{gradients}$$

The learning rate controls the size of the steps taken during each iteration of gradient descent. A smaller learning rate results in smaller, more precise steps, while a larger rate results in quicker but potentially less accurate updates.

```
cost = (1/(2*m)) * np.sum(errors**2)
```

The cost function is computed here as the mean squared error (MSE) between predicted and actual values:

$$J(\theta) = \frac{1}{2m} \sum (\hat{y} - y)^2$$

This cost function quantifies the error of the model, and gradient descent aims to minimize it by adjusting θ iteratively.

2.5.2 Hyperparameter Optimisation using Grid Search

Its hyperparameters were fine-tuned using the grid search method. The two hyperparameters optimised were learning rate and number of iterations. Learning rate dictates the size of steps that the algorithm takes in its goal to minimise the loss function. If it is too low, the algorithm can take too long to converge, whereas, if it is too large, the algorithm can overshoot the minimum. The number of iterations, if too high, implies unnecessary computation that can leave to overfitting, whereas, too few iterations can indicate a premature halting of the optimisation, failing to obtain optimal parameters.

2.6 Markov Decision Process (MDP)

To optimize students' performance, a simplified MDP was formulated, modelling the transition probabilities between students' actions and their resulting performance states. The objective was to identify an optimal policy that informs decision-making regarding study habits and social activities. The states represent different levels of academic performance, they are categorised as "bad" "OK" and "good" These designations reflect levels of student performance, with "bad" indicating low academic achievement, "OK" representing average performance, and "good" signifying high academic success.

The actions available to students encompass choices that can influence their academic performance. Specifically, the actions include "studytime" which pertains to the time allocated for studying; "dalc" representing the amount of alcohol consumed on work days; and "walc" which denotes weekend alcohol consumption.

The reward structure associated with each state is designed to provide incentive for higher performance. A reward of -10 was assigned to the "bad" state, therefore penalizing low achievement, while the "OK" state received a neutral reward of 0, and the "good" state was rewarded with a score of 10, reflecting the value placed on high performance.

Transition probabilities show the likelihood of moving from one state to another based on a specific action. These probabilities are defined by consulting the matrix of correlation favouring increasing studying time to drinking alcohol. For instance, when a student in the "bad" state opts for "studytime" instead of "walc" or "dalc" their probability of improving their performance is higher than that of choosing to drink alcohol.

The learning strategy employed in this MDP is value iteration, which iteratively computes the optimal value function and corresponding policy. Initially, the value function V for each state is set to zero. The algorithm then updates the expected value of taking each action for every state, taking into account the transition probabilities and associated rewards. This value update process continues until convergence is achieved, defined by a predetermined threshold θ.

The mathematical formulation for updating the value function is:

$$V(s) = \max_{a} \sum_{s'} P(s'|s,a) \left(R(s') + \gamma V(s') \right)$$

Here, $P(s'|s,a)$ represents the transition probability from state s to s' given action a, $R(s')$ denotes the reward for reaching state s', and γ is the discount factor that balances immediate and future rewards. For my implementation, the transition probabilities were not derived empirically, they were merely modelled from the correlation matrix. The correlation matrix provided a heuristic as to how the drinking alcohol on weekends / workdays would affect G3 scores and also how study time relates to G3 scores.

Upon convergence of the value function, the optimal policy is derived by selecting the action that maximizes the expected value for each state. Through this MDP framework, the project seeks to elucidate how different student actions can impact academic outcomes, thereby empowering students to make informed decisions that promote their academic success.

3 | Results and Discussion

3.1 Exploratory Data Analysis

3.1.1 Initial Check of Matrix

Below in 3.1 displays the initial check to familiarise oneself with the matrix providing information like the number of columns and some features.

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	4	0	11	11
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	2	9	11	11
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	6	12	13	12
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	0	14	14	14
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	0	11	13	13

5 rows × 33 columns

Figure 3.1: Data Set displayed using pandas.head()

3.1.2 Matrix of Correlation

The matrix below 3.2 displays a heatmap of correlation between numerical variables. The Pearson coefficient of correlation ranges from -1 to 1 denoting unit covariance.

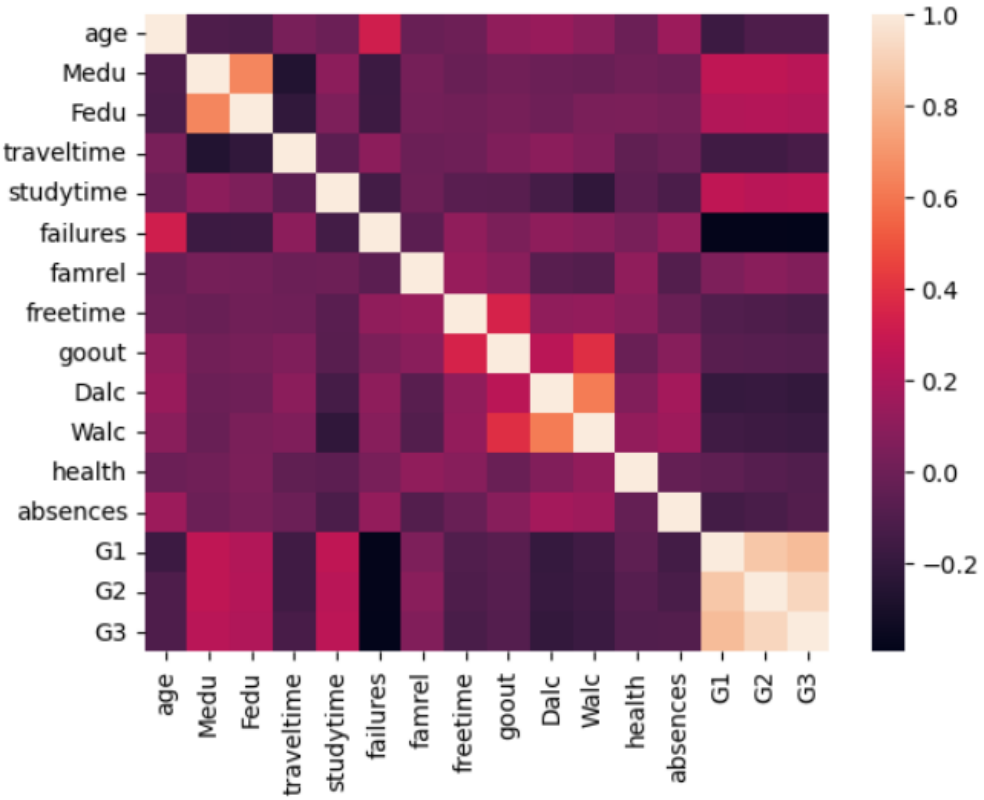

Figure 3.2: Correlation Matrix of Numerical Variables in Data Set

From this matrix, we can see that the previous exam results are more correlated with G3 than any other variables. This supports the conclusion drawn by Cortez and Silva [1] that the strongest predictors of student performance were past performances. They did however also determine that other factors can provide information in explaining exam result variance. It is not clear from this correlation matrix that any other variables are strongly correlated with G3, but it is observed that study time, failures, and the education level of students' parents appear to be somewhat correlated.

3.1.3 The Relationship Between Categorical Variables and G3 Scores

A table of t-test results was scores and analysed, the full table can be found in the appendix. Only features that had t-tests where the p value was below the Boneferri corrected α are displayed representing statistically significant results.

The following figure 3.3 shows the distributions of the categorical feature 'school' in relation to G3. *GS* refers to the school, *Gabriel Pereira*, whereas, *MS* refers to the school, *Mousinho da Silveira*.

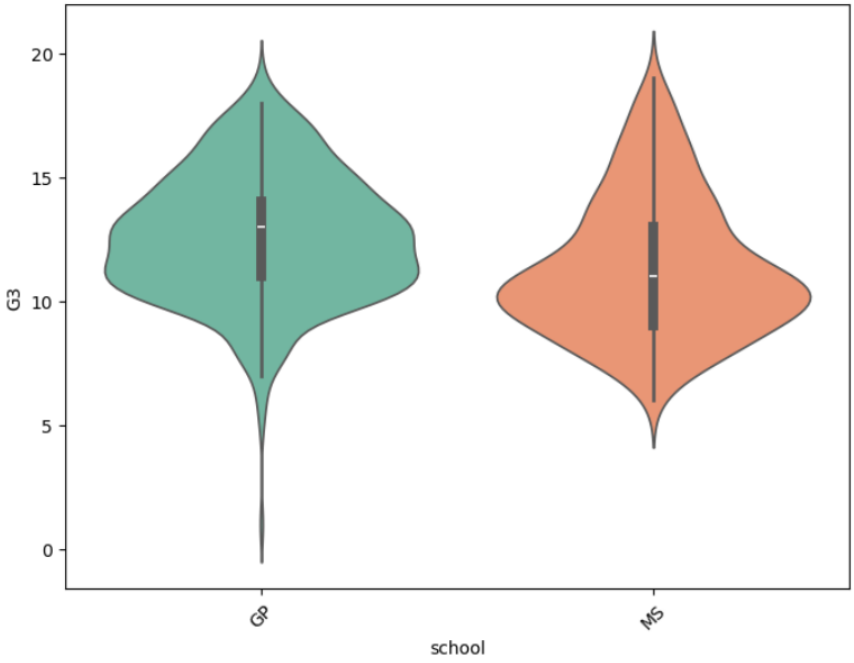

Figure 3.3: Comparing the G3 distribution between students attending GS and MS

This indicates a disparity in the student performances depending on the school. This could be a proxy for variables not measured in the data collection relating to the standard of teaching or performance of the school that the students attend.

Figure 3.4, displays the different distributions of G3 scores between students whose motivation for choosing their school differs.

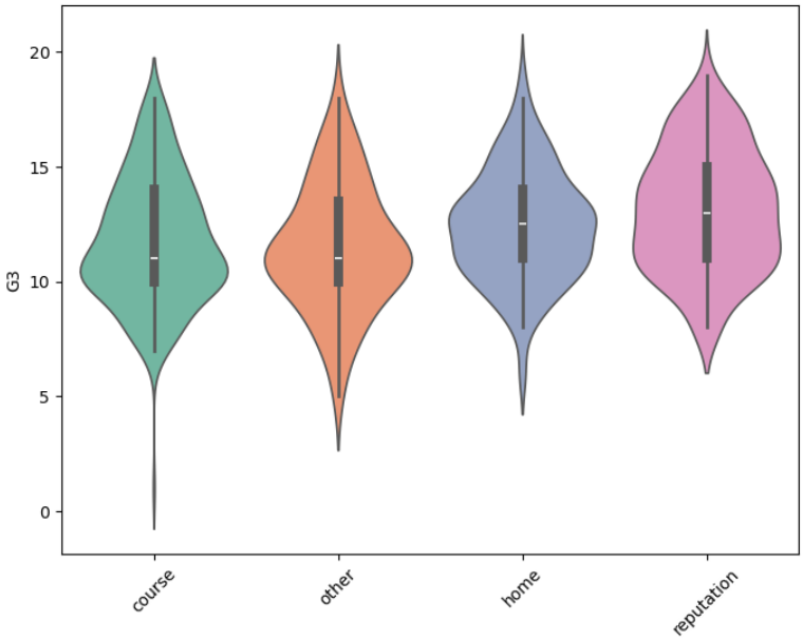

Figure 3.4: Comparing the G3 distribution between the different reasons students gave for choosing a school

This appears to follow on from the reasoning that the quality of school affects the students' exam performance. Intuitively, a school that fosters strong student performance would be one with a prestigious reputation. Thus, this choice may also reveal disparities between education quality across schools.

The following figure 3.5 displays a comparison of students' mothers' jobs in relation to G3 scores, where there is a significant difference in means between student' who's mothers are at home and those that are teachers.

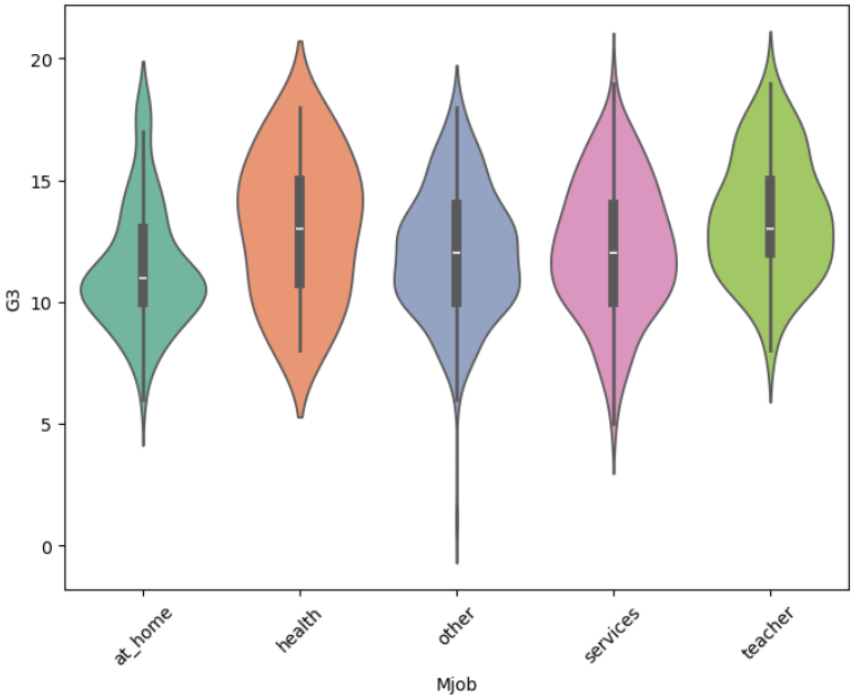

Figure 3.5: Comparing the G3 distribution between students' mothers' jobs

This further supports the designation of a link between the impact of a parent on the students' exam performance. As we saw with the numeric data, higher parental education is associated with higher exam performance. Therefore, the existence of a difference between the G3 results between students whose mothers do not work and those whose mothers are teachers supports this reasoning.

Figure 3.6 displays the disparity in G3 scores in student choice to continue to higher education.

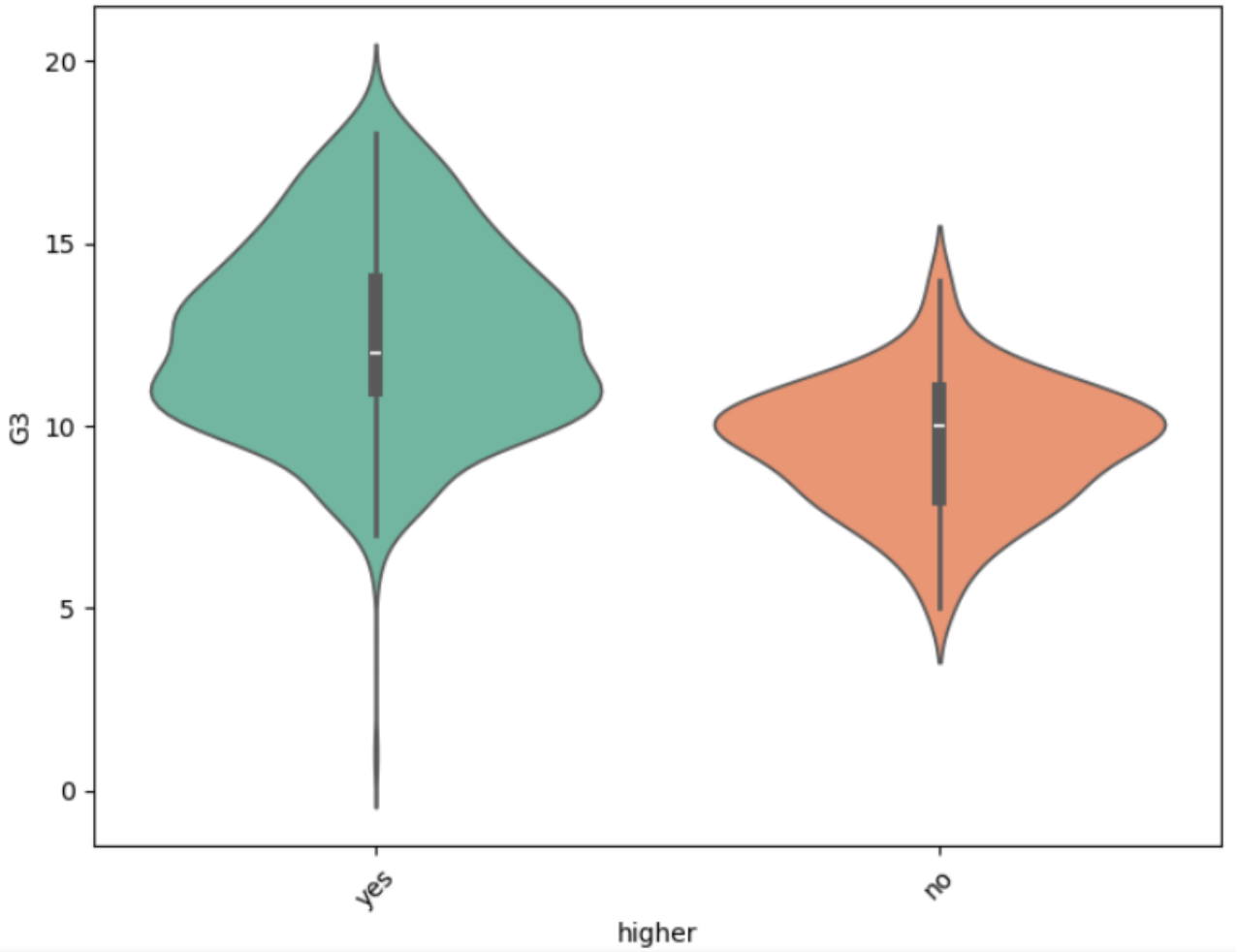

Figure 3.6: Comparing the G3 distribution between students intending to go to higher education ('yes') and students not intending to go to higher education ('no').

This better performance of students aiming to go to higher education can be contextualised, as most higher education institutions will take into consideration secondary level results for admissions purposes. These two are inherently related, but the direction of effect is not clear.

3.2 Principal Component Analysis

The following graph displayed in figure 3.7 displays the number of principal components and each of their explanatory power (explained variance ratio).

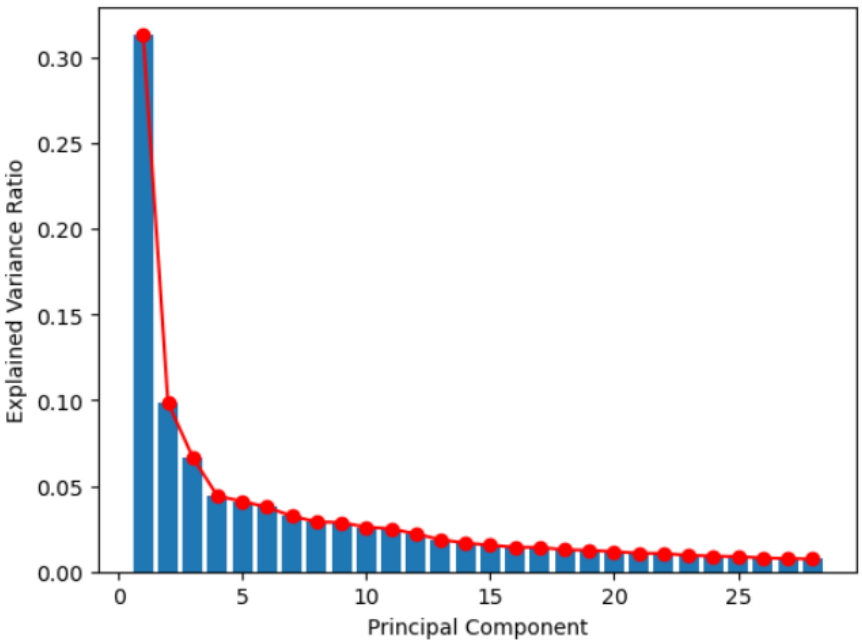

Figure 3.7: The Explained Variance Ratio of Each Principal Component

As we can see, there are 28 principal components that explain 95% of the variance of G3 scores. From this graph, we can see that the magnitude of the components tapers off quite quickly after the first 3 components. This implies a potential poor trade off between explanatory power and the number of dimensions.

3.3 Optimisation

3.3.1 Hyperparameter Refinement

The following two graphs, in figure 3.8, display the difference in cost with changes in hyperparameters.

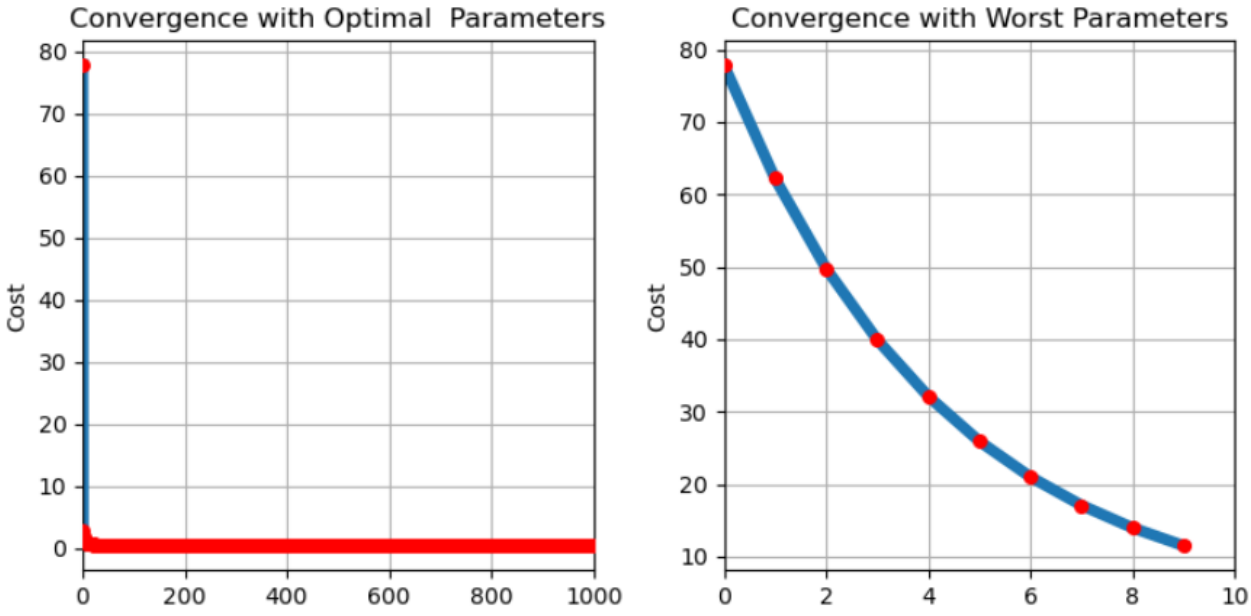

Figure 3.8: A plot of cost over iterations of the model using comparing the optimal and worst parameters identified using grid search

Here, it is noted that the model, leveraging the optimal parameters, reached a cost much closer to 0. However, it does so in N^3 of the worst parameters, where N is the number of iterations. It was also observed that the cost function tapers off quite quickly using the optimal parameters, drawing into question the number of parameters used.

3.3.2 Model Performance

The performance of a model can be measured in how well it can be used to predict the target variable, i.e., explain the variance of y. The $\sqrt{MSE}$ was chosen to replicate the measurement used by Cortez and Silva [1]. This represents the average magnitude of error by the model.

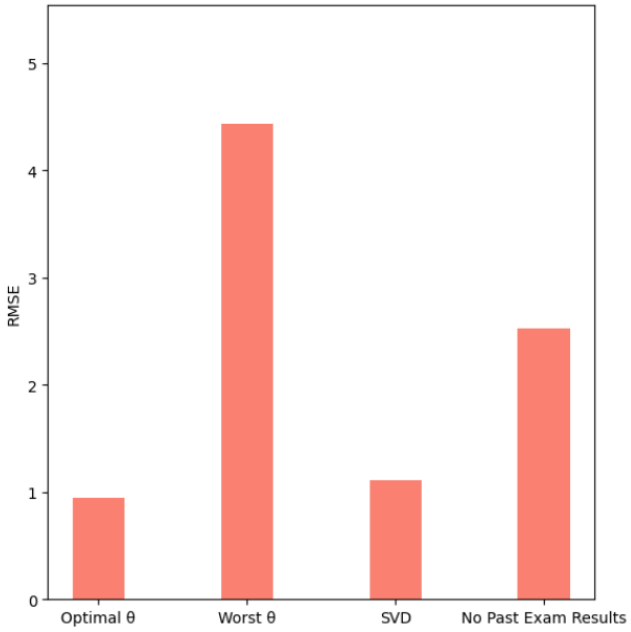

Figure 3.9: $\sqrt{MSE}$ Comparison Between Models

The SVD reduced model with approximately half the dimensions is still able to achieve comparable results to the model with the original data. The model without the presence of previous exam results ('G1' and 'G3') is outperformed by the model with these results included as features. Then, finally, the model that used the worst parameters as derived by grid search produces larger errors.

The following table 3.1 displays these $\sqrt{MSE}$ values.

Table 3.1: Root Mean Square Error (RMSE) for Different Models

Model	**RMSE**
Optimal θ	0.9450
Worst θ	4.4349
SVD	1.1079
No Past Exam Results	2.5264

Next, R^2 also referred to as the coefficient of determination, was measured across the 4 models. R^2 indicates percentage of variance of G3 explained by the model. The disparity in R^2 values are in line with the models' errors as seem with the $\sqrt{MSE}$ values.

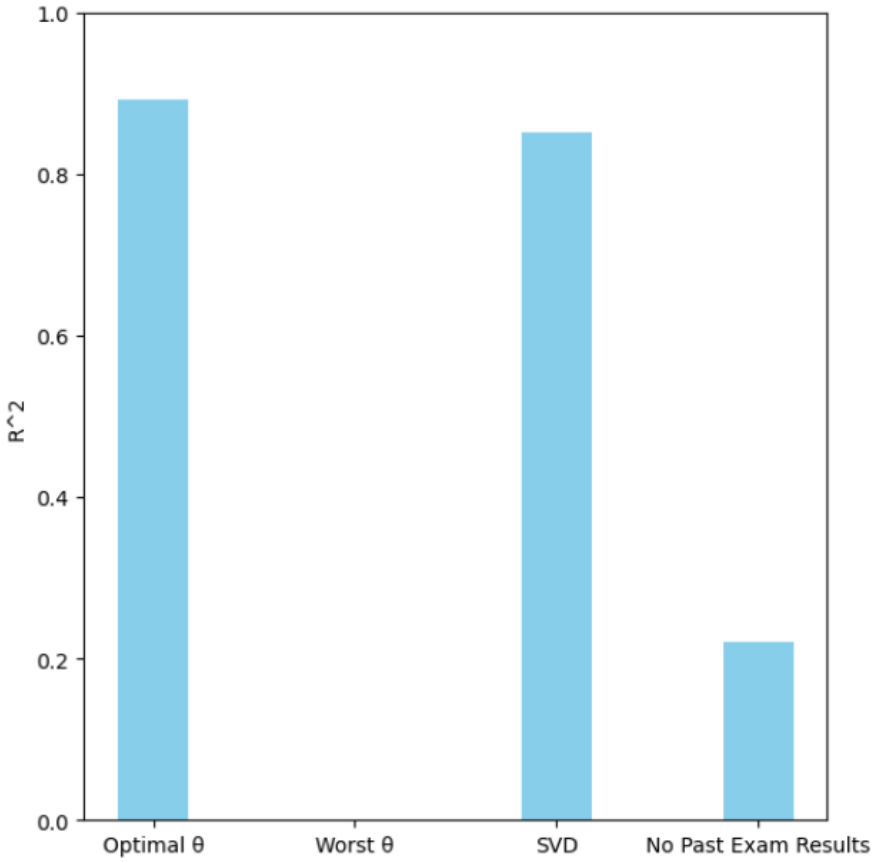

Figure 3.10: A Comparison of the R^2 Coefficient Between Models

Table 3.2: R-squared Values for Different Models

Model	**R-squared**
Optimal θ	0.8910
Worst θ	-1.3999
SVD	0.8502
No Past Exam Results	0.2212

3.4 Markov Decision Process

The following table 3.3, shows that the expected long term reward is the highest when students' performance is "good".

Table 3.3: Optimal Value Function

State	**Value**
bad	12.6714
OK	28.0541
good	36.1672

The optimal policy clearly shows us that increasing study time is always the best action to take for a student' when faced with other choices involving alcohol consumption relative to days of the week.

Table 3.4: Optimal Policy

State	**Action**
bad	studytime
ok	studytime
good	studytime

4 | Conclusion

This project successfully demonstrated the implementation of key artificial intelligence applications based on mathematical concepts. The balance between model accuracy and time complexity was highlighted with the number of components retained in the PCA, and the number of iterations when optimising parameters.

The ability of PCA leveraging the matrix factorization technique SVD to maintain model accuracy, all while reducing the number of model features, was demonstrated.

Results indicating the importance of previous exam results were shown using the correlation matrix.

A MDP was conceptualised to inform students of optimal choices to make using actionable data from the data set in order to improve results.

It was noticed that the choice of school and potentially the reputation of the school had a significant relationship with G3 results. This could be interesting to government informing them of potential education disparities between schools.

Finally, backed by both categorical and numerical data, the students' parents also impacted their G3 results.

5 | Word Count

Word count for report: 3220.

Bibliography

[1] Cortez, P. 2008, Student Performance, UCI Machine Learning Repository

A | Appendix

A.1 T-Test Table

Table A.1: Statistical T-Test Results Indicating Significance

Feature	Category 1	Category 2	T-Statistic	P-Value	Adjusted P-Value	Significant
school	GP	MS	5.6629	2.26E-08	9.49E-07	TRUE
sex	F	M	3.3429	0.0009	0.0369	FALSE
address	U	R	3.0976	0.0020	0.0856	FALSE
famsize	GT3	LE3	-0.4293	0.6679	28.0508	FALSE
Pstatus	A	T	0.1059	0.9157	38.4590	FALSE
Mjob	at_home	health	-3.7826	0.0002	0.0089	FALSE
Mjob	at_home	other	-2.3893	0.0174	0.7296	FALSE
Mjob	at_home	services	-2.6582	0.0083	0.3500	FALSE
Mjob	at_home	teacher	-5.7742	2.94E-08	1.23E-06	TRUE
Mjob	health	other	2.4280	0.0158	0.6626	FALSE
Mjob	health	services	1.7591	0.0802	3.3702	FALSE
Mjob	health	teacher	-0.9000	0.3700	15.5390	FALSE
Mjob	other	services	-0.6824	0.4954	20.8076	FALSE
Mjob	other	teacher	-4.2230	3.15E-05	0.0013	FALSE
Mjob	services	teacher	-3.2766	0.0012	0.0519	FALSE
Fjob	teacher	other	4.0449	6.30E-05	0.0026	FALSE
Fjob	teacher	services	3.8703	0.0001	0.0061	FALSE
Fjob	teacher	health	1.8892	0.0640	2.6899	FALSE
Fjob	teacher	at_home	3.7793	0.0003	0.0133	FALSE
Fjob	other	services	0.8036	0.4220	17.7245	FALSE
Fjob	other	health	-0.7390	0.4604	19.3359	FALSE
Fjob	other	at_home	1.0670	0.2866	12.0386	FALSE
Fjob	services	health	-0.9433	0.3467	14.5608	FALSE
Fjob	services	at_home	0.5158	0.6066	25.4755	FALSE
Fjob	health	at_home	1.1571	0.2517	10.5695	FALSE
reason	course	other	0.7061	0.4806	20.1841	FALSE
reason	course	home	-2.5386	0.0115	0.4825	FALSE
reason	course	reputation	-4.9322	1.17E-06	4.93E-05	TRUE
reason	other	home	-2.5455	0.0116	0.4884	FALSE
reason	other	reputation	-4.1301	5.27E-05	0.0022	FALSE
reason	home	reputation	-2.3334	0.0203	0.8535	FALSE
guardian	mother	father	-0.9933	0.3209	13.4797	FALSE
guardian	mother	other	2.3039	0.0217	0.9094	FALSE
guardian	father	other	2.7749	0.0061	0.2554	FALSE
schoolsup	yes	no	-2.3881	0.0172	0.7237	FALSE
famsup	no	yes	0.0897	0.9285	38.9977	FALSE
paid	no	yes	1.6264	0.1044	4.3832	FALSE
activities	no	yes	-2.1130	0.0350	1.4697	FALSE
nursery	yes	no	1.3188	0.1877	7.8834	FALSE
higher	yes	no	8.9918	2.80E-18	1.18E-16	TRUE
internet	no	yes	-3.1932	0.0015	0.0620	FALSE
romantic	no	yes	1.5104	0.1314	5.5208	FALSE

A.2 Code

```
1
2  import pandas as pd
3  import os
4  import matplotlib.pyplot as plt
5  import seaborn as sns
6  import numpy as np
7  from scipy import stats
8  from sklearn.preprocessing import StandardScaler
9  from sklearn.model_selection import train_test_split
10 from sklearn.linear_model import LinearRegression
11 from sklearn.metrics import mean_squared_error, r2_score
12 from sklearn.tree import DecisionTreeClassifier
13 from sklearn.metrics import accuracy_score
14 from sklearn.preprocessing import OneHotEncoder
15 from sklearn.compose import ColumnTransformer
16 from sklearn.preprocessing import StandardScaler
17 from sklearn.compose import make_column_selector
18 from sklearn.metrics import mean_squared_error
19
20
21 # Read data
22 data = pd.read_csv('C:/Users/josep/Downloads/student-por.csv')
23 # Overview of data
24 data.head()
25
26 # Correlation Matrix of variables.
27 dataplot = sns.heatmap(data.corr(numeric_only=True))
28 plt.show()
29
30 sns.countplot(x='G3', data=data)
31 # outliers at 0, possible error
32 # Get rid of rows where G3 is 0.
33 data = data[data['G3'] != 0]
34 #sns.countplot(x='G3', data=data)
35 import pandas as pd
36 import numpy as np
37 import seaborn as sns
38 from scipy import stats
39 import matplotlib.pyplot as plt
40
41
42 results = []
43 num_tests = 0  # To count the number of pairwise comparisons
44
45 # For each categorical feature
46 for feature in categorical_columns:
47     plt.figure(figsize=(8, 6))
48     sns.violinplot(x=feature, y='G3', data=data, palette='Set2', hue=feature)
49     #plt.title(f'Distribution of Categories in {feature} in relation to G3')
50     plt.xticks(rotation=45)
51
52     # Get unique categories for the feature
53     categories = data[feature].unique()
54
55     for i in range(len(categories)):
56         for j in range(i+1, len(categories)):
57             group1 = data[data[feature] == categories[i]]['G3']
58             group2 = data[data[feature] == categories[j]]['G3']
59
60             # Perform t-test between the two groups
61             t_stat, p_val = stats.ttest_ind(group1, group2)
62             num_tests += 1  # Increment number of tests
```

```
63
64              results.append([feature, categories[i], categories[j], t_stat, p_val])
65
66  # Convert results into a DataFrame
67  results_df = pd.DataFrame(results, columns=['Feature', 'Category 1', 'Category 2', '
        T-Statistic', 'P-Value'])
68
69  # Bonferroni correction
70  bonferroni_alpha = 0.05 / num_tests  # Adjusted alpha
71  results_df['Adjusted P-Value'] = results_df['P-Value'] * num_tests
72
73  results_df['Significant'] = results_df['Adjusted P-Value'] < bonferroni_alpha
74
75  results_df.to_csv('categorical_ttests_with_bonferroni.csv', index=False)
76
77  results_df
78  # Separate features and target variable
79  X = data.drop("G3", axis=1)
80  y = data["G3"]
81  # Data types of features
82  #X.info()
83  # Inspect easily the  presence of missing values.
84  # print(data.isnull().sum())
85  # Use ColumnTransformer to define transformer that one hot encodes categorical
        features and standardizes numerical features
86  col_trans = ColumnTransformer(
87      [
88          ('one_hot', OneHotEncoder(), make_column_selector(dtype_include=object)),
89          ('standardize', StandardScaler(), make_column_selector(dtype_include=np.
                number))
90      ]
91  )
92  X_scaled_and_encoded = col_trans.fit_transform(X)
93  #convert back to df
94  X_scaled_and_encoded = pd.DataFrame(X_scaled_and_encoded, columns=col_trans.
        get_feature_names_out())
95
96  # SVD matrix
97  U, s, Vh = np.linalg.svd(X_scaled_and_encoded)
98
99  # Explained variance ratio (normalized singular values)
100 explained_variance = (s ** 2) / np.sum(s ** 2)
101 cumulative_explained_variance = np.cumsum(explained_variance)
102
103 # Determine the number of components that explain > 95% variance
104 num_components = np.argmax(cumulative_explained_variance >= 0.95) + 1
105 Vh_reduced = Vh[:num_components, :]
106
107 # Reconstruction
108 X_svd = X_scaled_and_encoded.dot(Vh_reduced.T)
109
110 # Compute the importance of each component (more variance explained: more important)
111 explained_variance_ratio = (s ** 2) / np.sum(s ** 2)
112
113 # Visualise explained variance ratios
114 plt.bar(range(1, num_components + 1), explained_variance_ratio[: num_components],
        align="center")
115 plt.plot(range(1, num_components + 1), explained_variance_ratio[: num_components],
        color='red', marker='o', linestyle='-', label='Line Plot')
116
117 plt.xlabel("Principal Component")
118 plt.ylabel("Explained Variance Ratio")
119 #plt.title("Explained Variance Ratio per Principal Component (SVD)")
120 plt.show()
121 # Implementing gradient descent for parameter optimisation
```

```
def add_beta_init_theta(X):
    m = len(y)
    # Add beta (y offset)
    X = np.hstack((np.ones((m, 1)), X_scaled_and_encoded))
    # Initialize theta
    theta = np.zeros(X.shape[1])
    return X, theta

# Define the gradient descent function, from turorial 10-2
def gradient_descent(X, y, theta, learning_rate, iterations):
    m = len(y)
    cost_history = []

    for i in range(iterations):
        predictions = X.dot(theta)
        errors = predictions - y
        gradients = (1/m) * X.T.dot(errors)
        theta -= learning_rate * gradients
        cost = (1/(2*m)) * np.sum(errors**2)
        cost_history.append(cost)

    return theta, cost_history
    def plot_costs(costs, titles):
    num_plots = len(costs)
    rows = 2
    cols = 2
    fig, axes = plt.subplots(rows, cols, figsize=(8, 8))

    # Flatten the axes array for easy iteration
    axes = axes.flatten()

    for ax, cost_history, title in zip(axes, costs, titles):
        ax.plot(range(len(cost_history)), cost_history, linewidth=5)
        ax.set_ylabel("Cost")
        ax.set_title(f"Convergence with {title}")
        ax.grid(True)
        ax.set_aspect('auto')  # Same sizes
        ax.set_xlim([0, len(cost_history)])
        ax.scatter(range(len(cost_history)), cost_history, color='red', s=30, zorder
            =5)  # Adjust 's' for circle size


    # Set equal size for all subplots
    plt.subplots_adjust(hspace=0.4, wspace=0.4)  # Adjust spacing between plots
    plt.xlabel("Iterations")
    plt.tight_layout()
    plt.show()
    def grid_search(X, y, theta, lr_list, it_list):
    # Initialise min cost
    min_cost = 1
    opt_it = -1
    opt_lr = -1
    # Test all combinations of learning rates
    for lr in lr_list:
        print(f'lr: {lr}')
        # And number of iterations
        for n_it in it_list:
            print(f'n_it: {n_it}')
            # Using gradient descent
            theta, cost_history = gradient_descent(X_grad_og, y, theta, lr, n_it)
            # That minimise cost
            cur_cost = cost_history[-1]
            print(cur_cost)
            if cur_cost < min_cost:
                opt_it = n_it
```

```
                opt_lr = lr
                min_cost = cur_cost

    return opt_it, opt_lr


learning_rate_vals = [0.01, 0.1]
num_iteration_vals = [10, 100, 1000]

X_grad_og, theta = add_beta_init_theta(X_scaled_and_encoded)
optimal_num_iterations, optimal_learning_rate = grid_search(X_grad_og, y,theta,
    learning_rate_vals, num_iteration_vals)
print(f'lr: {optimal_learning_rate} || it: {optimal_num_iterations}')
    # Function for adding bias term and initializing theta
def add_beta_init_theta(X):
    # Add a bias term (column of ones) to the feature matrix X
    X_with_bias = np.c_[np.ones((X.shape[0], 1)), X]
    # Initialize theta (weights) to zeros
    theta = np.zeros(X_with_bias.shape[1])
    return X_with_bias, theta

# Function for Gradient Descent with train/test split
def evaluate_gradient_descent(X, y, learning_rate, num_iterations):
    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
        random_state=42)

    # Initialize theta and add bias term to training and testing sets
    X_train_grad, theta = add_beta_init_theta(X_train)
    X_test_grad, _ = add_beta_init_theta(X_test)  # Ensure same bias term for test
        data

    # Perform gradient descent
    theta, cost_history = gradient_descent(X_train_grad, y_train, theta,
        learning_rate, num_iterations)

    # Predict on test data
    predictions = X_test_grad.dot(theta)
    # Evaluate MSE
    mse = mean_squared_error(y_test, predictions)
    # Get root MSE for more interperatable results.
    rmse = np.sqrt(mse)
    # Evaluate R^2
    R2 = r2_score(y_test, predictions)


    return rmse, R2, cost_history

X_x = X_scaled_and_encoded.drop(["standardize__G1", "standardize__G2"], axis=1)
# Evaluate using gradient descent
rmse_og, R2_og, cost_og = evaluate_gradient_descent(X_scaled_and_encoded, y,
    optimal_learning_rate, optimal_num_iterations)
rmse_svd, R2_svd, cost_svd = evaluate_gradient_descent(X_svd, y,
    optimal_learning_rate, optimal_num_iterations)
rmse_x, R2_x, cost_x = evaluate_gradient_descent(X_x, y, optimal_learning_rate,
    optimal_num_iterations)
rmse_grad_bad, R2__grad_bad, cost__grad_bad = evaluate_gradient_descent(
    X_scaled_and_encoded, y, 0.01, 10)




# Output the results
print(f"Optimal        - RMSE: {rmse_og}, R^2: {R2_og}")
print(f"Worst         - RMSE: {rmse_grad_bad},    R^2: {R2__grad_bad}")
```

```
print(f"SVD                    - RMSE: {rmse_svd},  R^2: {R2_svd}")
print(f"No Past Exam Results - RMSE: {rmse_x},    R^2: {R2_x} ")


r_squares = [R2_og, R2__grad_bad, R2_svd, R2_x]
rmse_vals = [rmse_og, rmse_grad_bad, rmse_svd, rmse_x]
costs = [cost_og, cost__grad_bad, cost_svd, cost_x]
titles = ["Optimal   ", "Worst   ", "SVD", "No Past Exam Results"]

plot_costs(costs, titles)
# Plot RMSE and R^2
# Plot RMSE and R^2
def plot_r2_rmse(titles, r2vals, rmse_vals):
    x = np.arange(len(titles))  # The label locations
    width = 0.35  # The width of the bars

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

    # Plot for R^2
    ax1.bar(x, r2vals, color='skyblue', width=width)
    ax1.set_ylabel('R^2')
    #ax1.set_title('R^2 Scores by Model')
    ax1.set_xticks(x)
    ax1.set_xticklabels(titles)
    ax1.set_ylim(0, 1)

    # Plot for RMSE
    ax2.bar(x, rmse_vals, color='salmon', width=width)
    ax2.set_ylabel('RMSE')
    #ax2.set_title('RMSE Scores by Model')
    ax2.set_xticks(x)
    ax2.set_xticklabels(titles)
    ax2.set_ylim(0,max(rmse_vals) * 1.25)

    plt.tight_layout()
    plt.show()
    import numpy as np

states = ['bad', 'ok', 'good'] # Student performance
actions = ['studytime', 'dalc', 'walc'] # Actions that students can take.

rewards = {
    'bad': -10,
    'ok': 0,
    'good': 10
}

transition_probs = {
    'bad': {
        'studytime': {'bad': 0.6, 'ok': 0.3, 'good': 0.1},
        'dalc': {'bad': 0.8, 'ok': 0.15, 'good': 0.05},
        'walc': {'bad': 0.7, 'ok': 0.2, 'good': 0.1}
    },
    'ok': {
        'studytime': {'bad': 0.1, 'ok': 0.6, 'good': 0.3},
        'dalc': {'bad': 0.4, 'ok': 0.5, 'good': 0.1},
        'walc': {'bad': 0.5, 'ok': 0.4, 'good': 0.1}
    },
    'good': {
        'studytime': {'bad': 0.05, 'ok': 0.25, 'good': 0.7},
        'dalc': {'bad': 0.1, 'ok': 0.3, 'good': 0.6},
        'walc': {'bad': 0.2, 'ok': 0.4, 'good': 0.4}
    }
}

```

```
gamma = 0.9  # Discount factor
theta = 0.001  # Convergence threshold

#  Value function
V = {state: 0 for state in states}

def value_iteration():
    while True:
        delta = 0
        for state in states:
            v = V[state]
            # Compute the value of taking each action and update V[state]
            V[state] = max(
                sum(transition_probs[state][action][next_state] *
                    (rewards[next_state] + gamma * V[next_state])
                    for next_state in states)
                for action in actions
            )
            delta = max(delta, abs(v - V[state]))
        if delta < theta:
            break

value_iteration()

# Optimal policy
policy = {}
for state in states:
    best_action = None
    best_value = float('-inf')
    for action in actions:
        action_value = sum(transition_probs[state][action][next_state] *
                           (rewards[next_state] + gamma * V[next_state])
                           for next_state in states)
        if action_value > best_value:
            best_value = action_value
            best_action = action
    policy[state] = best_action

print("Optimal Value Function:", V)
print("Optimal Policy:", policy)




plot_r2_rmse(titles, r_squares, rmse_vals)
```

Listing A.1: Column Transformation in Python