Aalto University

CS-C3240 - Machine Learning

Predicting exam score based on hours studied, attendance and previous exam score

Date: October 4, 2024

1 Introduction

The goal of this paper is to predict a person's final exam score based on hours spent studying, previous exam score, attendance percentage, motivation level and average slept hours. Based on our own experience, we except these criteria to predict the exam score quite accurately. The motivation of this paper is to show that these factors have a positive effect on the exam score. If the model turns out to be accurate, future success in exams can be predicted using this model. For example a student could predict his score before an exam and based on that decide if he needs to spend more time on that course.

In section 2 the features and labels are explained. Section 3 discusses the two machine learning methods used, which are linear regression and multilayer perceptron. Section 4 goes over the results and section 5 discusses the conclusions based on the results.

2 Problem formulation

This paper will predict the final exam score based on different features. The data set used is *Student Performance Factors* from kaggle.com [1]. The data points represent a particular students statistics in twenty different features, but this paper focuses on five features that are usually attributed to good grades. The data set has 6607 entries. The used features are:

- The effect of the percentage of attendance
- The hours spent studying
- The average amount of sleep per night
- The previous exam score
- The motivation level

The first four out of those are continuous, while the motivation level is categorized into three categories: low, medium and high. These can be represented with the numbers 1, 2 and 3. Hours studied varies from 1 to 44 hours, attendance percentage from 60 to 100 percentage, sleeping hours from 4 to 10 hours, and the previous exam score ranges from 55 to 100 points. The attendance is the percentage of the lectures that a student has attended. Sleeping hours is the average hours of sleep per night. The problem in this paper is a supervised learning problem, with the label being the exam score, which ranges from 55 to 101 points.

3 Methods

In this paper the used models are linear regression and multi-layer perceptron. The choice of linear regression was made based on the scatter visualisation where there is a clear linear relationship between some features and the label. The linear relationship of attendance, previous exam scores and studying hours can be seen in Figure 1 as well as the lack of it for sleeping hours and motivation level. In Figure 2 it can also be seen that hours slept and motivation have little to none correlation with exam score. As a result, those two will be discarded from the dataset.

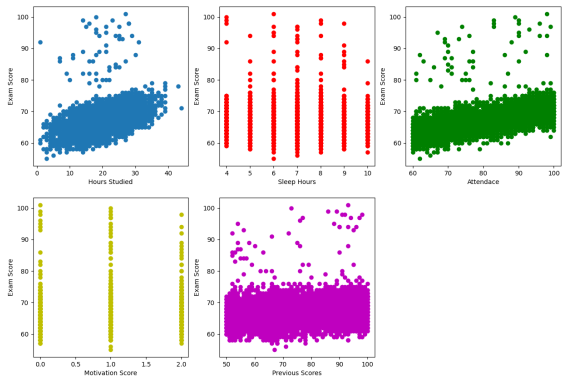

Figure 1: Relationship between features and label

Figure 2: Correlations between features

The choice of multi-layer perceptron, on the other hand, was made on the basis that it is generally good at detecting non-linear patterns. We think that by comparing these models, we are able to get a more thorough understanding of the data and train an accurate model for it. We have decided to use five neurons and 1,2,4,6,8,10 and 11 layers, so that our model is not too heavy but still accurate.

In this paper the mean squared error will be used as the loss function for linear regression as it is recommend on the course page. It is also simple, accurate and has ready made libraries, which means that it was easy to implement. For multi-layer perceptron logistic loss was chosen as the loss function. It was chosen because it is already contained in scikit MPLClassifier which is a multi-layer perceptron model.

In our data the students are not placed in any particular order, and therefore there is no bias what data is in which set. We have decided to go with a 60:20:20 split between the training, validation and testing data so that we have sufficient amount of training data to make the model precise, and still have enough data for validation and testing. The training, validation and testing datasets were created using the command: train_test_split.

4 Results

The results for linear regression can be seen in Figure 3 and for MLP in Figure4.

```
Training error: 6.544509761269392
Prediction error for validation: 6.267276374383942
Prediction error for test: 6.267276374383942
```

Figure 3: LinearRegression mean squared error

	mlp_train_errors	mlp_val_errors
1 layer	6.739136	6.485401
2 layers	6.606319	6.333248
4 layers	6.550761	6.282033
6 layers	6.576878	6.294922
8 layers	6.588113	6.340613
10 layers	6.650974	6.381846
11 layers	6.602430	6.349792

Figure 4: MLP regression errors for different number of layers

In figure 4 it can be perceived that the best fit for MLP is with 4 layer, and

at 6 layers the model clearly starts overfitting. With 4 layers, the training error is 6.55 and a squared validation error is 6.28. From figure 3 it can be seen that the for linear regression the training error is 6.54 and the squared validation error is 6.26. Therefore, it is clear that linear regression suitable for our application and the neural network doesn't provide any benefit, which was expected as the variables seemed to have a linear relationship. Linear regression also has a lower validation error.

By this basis, we chose linear regression as our final model. Its training error for the dataset was 6.54 and validation error 6.26. Since the test results, our label, spans from 55 to 101, a squared error of 6.26 can be considered acceptable. The difference between our training error and validation error is marginal which indicates that our model is not overfitting. We used the remaining 20% share of our dataset for testing as explained in chapter 3. This was done using train_test_split command. For linear regression we got a test error of 6.26. This is a very good sign as it shows that our model doesn't overfit.

5 Conclusion

In this report we wanted to predict a pupil's exam score based on 5 features. These were percentage of lectures attended, hours spent studying, average slept hours, previous exam score and motivation level. It was found out using correlation matrices and graphs that only three of these are significant factors. Thus we reduced our features to hour studied, attendance percentage and previous exam score.

We split our dataset into 60% training, 20% validation and 20% testing data. We used linear regression and Multi-layer Perception as our machine learning methods. From these two we chose linear regression as our final model, as there was no significant advantage in the more complex MLP. With our model, the training error was 6.54 and the validation error 6.26. These two values are in near proximity, which indicates that the model is not overfitting. A validation error of 6.54 is suitable, as our label's values scale form 55 to 101.

Thus it can be concluded that we can pretty accurately predict a pupil's final exam score based on the chosen factors. In the future, it could be considered to add more features to the model to improve the accuracy. In that case, other methods could work better, for example polynomial regression, as it is expected that not all features are linearly dependant. We could also increase the size of the dataset as now it has only about 6000 entries.

References

[1] Student Performance Factors *Kaggle Datasets*. Retrieved from https://www.kaggle.com/datasets/lainguyn123/student-performance-factors.

6 Appendix

```
%config Completer.use_jedi = False
# enable code auto-completion
# needed imports
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns #data visualization library
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
MSE = mean_squared_error

#import the data from the .csv
df = pd.read_csv('StudentPerformanceFactors.csv')

#check the shape
print(df.shape)

#select the needed columns
df = df[['Hours_Studied', 'Attendance', 'Sleep_Hours',
    'Exam_Score', 'Motivation_Level','Previous_Scores']]
print(df.head(5))

#put them into different variables for plotting
hours = df['Hours_Studied'].to_numpy()
att = df['Attendance'].to_numpy()
sleep = df['Sleep_Hours'].to_numpy()
exam = df['Exam_Score'].to_numpy()
moti = df['Motivation_Level'].to_numpy()
```

```
prev = df['Previous_Scores'].to_numpy()
print(moti.shape)

#let 0 be low, 1 be medium and 2 high motivation
#make the motivation levels numbered

nmoti =[]

for m in moti:
    if m == 'Low':
        nmoti.append(0)
    elif m == 'Medium':
        nmoti.append(1)
    else:
        nmoti.append(2)

print(nmoti[0:5])

#plot them all in subplots
fig, axes = plt.subplots(2,3, figsize=(15,10))
axes[0,0].scatter(hours, exam)
axes[0,0].set_ylabel('Exam Score')
axes[0,0].set_xlabel('Hours Studied')

axes[0,1].scatter(sleep, exam, c='r')
axes[0,1].set_ylabel('Exam Score')
axes[0,1].set_xlabel('Sleep Hours')

axes[0,2].scatter(att, exam, c='g')
axes[0,2].set_ylabel('Exam Score')
axes[0,2].set_xlabel('Attendace')

axes[1,0].scatter(nmoti, exam, c='y')
axes[1,0].set_ylabel('Exam Score')
axes[1,0].set_xlabel('Motivation Score')

axes[1,1].scatter(prev, exam, c='m')
axes[1,1].set_ylabel('Exam Score')
axes[1,1].set_xlabel('Previous Scores')

fig.delaxes(axes[1,2])

#save it
fig.savefig('effects.eps')
```

```
#calculate the correlation matrix
cr_df = df.drop(columns=['Motivation_Level'])
cr_df['Motivation_numerated'] = nmoti
correlation_matrix = cr_df.corr()

#create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
    vmin=-1, vmax=1, center=0)
plt.title('Correlation Matrix of Student Performance Features')
plt.tight_layout()

#save it
plt.savefig('corr.eps')
plt.show()
# If you want to focus specifically on correlations
# with Exam_Score
exam_score_correlations = correlation_matrix[
    'Exam_Score'].sort_values(ascending=False)
print("Correlations with Exam Score:")
print(exam_score_correlations)

#code for fitting linear regression
#first we need to split our dataset into
#training and validation
X = cr_df.drop(columns=['Exam_Score', 'Sleep_Hours',
    'Motivation_numerated']).to_numpy()
y = cr_df['Exam_Score'].to_numpy()
regr = LinearRegression()
#X_train, X_val, y_train, y_val = ...
    train_test_split(X, y, test_size = 0.3,random_state=1)
X_split, X_test, y_split, y_test = train_test_split(
    X, y, test_size = 0.2,train_size = 0.8,random_state=1)
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size = 0.25,train_size=0.75,random_state=1)
regr.fit(X_train, y_train)
y_pred_train = regr.predict(X_train)
train_error = MSE(y_train, y_pred_train)
print('Training error:',train_error)
y_pred_val = regr.predict(X_val)
pred_error = MSE(y_val, y_pred_val)
print('Prediction error for validation:',pred_error)
y_pred_test = rgr.predict(X_test)
```

```
test_error = MSE(y_test, y_pred_test)
print('Prediction error for test:',pred_error)


from sklearn.neural_network import MLPRegressor

# We will use the same code as in this courses assignment 3
# but we tweak the number of layers and neurons
# code for neural network approach

## define a list of values for the number of hidden layers
num_layers = [1,2,4,6,8,10,11]    # number of hidden layers
num_neurons = 5  # number of neurons in each layer

mlp_tr_errors = []
mlp_val_errors = []

for i, num in enumerate(num_layers):
    hidden_layer_sizes = tuple([num_neurons]*num)

    mlp_regr = MLPRegressor(
        max_iter = 5000,random_state = 1,
        hidden_layer_sizes = hidden_layer_sizes)
    mlp_regr.fit(X_train,y_train)


    y_pred_train = mlp_regr.predict(X_train)
    tr_error = mean_squared_error(y_train, y_pred_train)
    y_pred_val = mlp_regr.predict(X_val)
    val_error = mean_squared_error(y_val, y_pred_val)

    mlp_tr_errors.append(tr_error)
    mlp_val_errors.append(val_error)

print(mlp_tr_errors)
print(mlp_val_errors)

# plot the training errors

plt.figure(figsize=(8, 6))

plt.plot(num_layers, mlp_tr_errors, label = 'Train')
plt.plot(num_layers, mlp_val_errors,label = 'Valid')
plt.xticks(num_layers)
```

```
plt.legend(loc = 'upper left')

plt.xlabel('Layers')
plt.ylabel('Loss')
plt.title('Train vs validation loss')
plt.show()

m_errors = {"mlp_train_errors":mlp_tr_errors,
    "mlp_val_errors":mlp_val_errors}
pd.DataFrame(m_errors).rename(index={0: "1 layer",
    1: "2 layers", 2: "4 layers",3:"6 layers",4:"8 layers",
    5:"10 layers",6:"11 layers"})
```