

**MACHINE LEARNING - IT4060**

**ASSIGNMENT 02**

B.Sc. (Hons) Degree in Information Technology

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

**Group Members**

|  |  |
| --- | --- |
| **Student ID** | **Name** |
| IT18149654 | Rajapaksha T.N. |
| IT18148350 | Naidabadu N.I. |
| IT18149272 | Perera M.J.F.R. |

Problem

Education is a vitally important aspect as it is directly linked with the social and economic development of any country. Countries that contain students with high education levels, achieve their social and economic goals smoothly. Students’ academic performance is playing an important role in education in any school or institution. An institution that has students with high academic performances, produces quality graduates to the country.

Student performance can be measured by examinations which is a process of testing the abilities and achievements of students. The student’s performance at examinations is affected by several reasons such as family relationship status, parent's cohabitation status, support from the school, extra-curricular activities, and student’s alcohol usage. Many students have low performance regarding their academic studies due to not fulfilling the above-mentioned factors.

Due to not achieving performance goals, students are facing many issues regarding their professional careers such as low qualifications for applying for jobs, unemployment, lack of performance, low salary, and low progress on the career path. Furthermore, unemployment is affecting the economic development of countries and causes many social issues as well. So that, enhancing the student’s performance to achieve educational goals will help to construct a better economic and social environment for a country.

If there is a way to identify the factors that decrease the performance and rectify them, it will be very supportive for the students to achieve higher performances in their academic studies. We are going to address the problem of the low performance of Portuguese students in examinations. Even though the Portuguese population has improved its educational levels within the last few decades, still Portugal exists in Europe’s tail end. The main reason for this problem is lack of performance in Mathematics and Portuguese language. So that, there is a crucial need to improve the Portuguese student’s performance level on these subjects.

Our goal is to predict the student performance against different factors such as family relationship status, parent's cohabitation status, support from the school, extra-curricular activities, and student’s alcohol usage which affect the students’ performance using Machine Learning (ML) approaches. Then we can explore the vital factors which are affecting the student’s performance in Mathematics and the Portuguese language and improve them. Further, this model will be helpful to identify struggling students who may need more attention and help. This will be very helpful for the students as well as educators to improve the students’ performance levels, minimize the failure rates of students, and improve the quality of education in Portugal.

Dataset

Background

Link: <https://archive.ics.uci.edu/ml/datasets/Student+Performance>

The dataset which we are going to use is the **student performance dataset** which is hosted on UC Irvine Machine Learning Repository. The dataset was donated on the 27th of November 2014 by Paulo Cortez and Alize Silva.

The student performance dataset contains student achievements in secondary education of two Portuguese schools. This dataset contains 33 attributes which categorize into student family background, school-related features, demographical data, social behavior-related data, grades etcetera.

There are two datasets regarding student performances for subjects: mathematics (student-mat.csv) and Portuguese language (student-por.csv). The mathematics performance dataset contains 395 records, and the Portuguese language performance dataset contains 649 records. Thus, the combined dataset contains 1044 records. The data collection is based on school reports and questionnaires.

Attributes

The attributes of the dataset, possible values, and their datatypes are briefly discussed below.

* sex
* This attribute refers to the gender of the student. The data type of this attribute is binary. “F” indicates a female student and “M” indicates a male student.
* age
* This attribute refers to the student's age. The data type is numeric which varies from 15 to 22.
* address
* This attribute refers to the student's home address type which is either urban or rural. The data type is binary which are either “U” - urban or “R” - rural.
* famsize
* This attribute refers to the family size of the student. The data type of this attribute is binary which are either “LE3” - less or equal to 3 or “GT3” - greater than 3.
* Pstatus
* This attribute refers to the parent's cohabitation status of the student. The data type of this attribute is binary which are either “T” being living together and “A” being apart from each other.
* Medu
* This attribute refers to the mother's education level of the student. The data type of this attribute is numeric which varies from 0 to 4. 0 - none, 1 - primary education (up to 4th grade), 2 – 5th to 9th grade, 3 – secondary education, or 4 – higher education
* Fedu
* This attribute refers to the father's education level of the student. The data type of this attribute is numeric which varies from 0 to 4. 0 - none, 1 - primary education (up to 4th grade), 2 – 5th to 9th grade, 3 – secondary education, or 4 – higher education
* Mjob
* This attribute refers to the mother's profession of the student. The data type of this attribute is nominal. Professions include “teacher”, “health”, “services”, “at\_home” or “other”.
* Fjob
* This attribute refers to the father's profession of the student. The data type of this attribute is nominal. Professions include “teacher”, “health”, “services”, “at\_home” or “other”.
* guardian
* This attribute refers to the student's guardian status. The data type of this attribute is nominal which includes the values of “mother”, “father”, or “other”.
* traveltime
* This attribute refers to the home-to-school travel time of the student. The data type of this attribute is numeric. Values vary from 1 to 4. 1 – less than 15 minutes, 2 - 15 to 30 minutes, 3 - 30 minutes to 1 hour, or 4 – more than 1 hour.
* studytime
* This attribute refers to the weekly study time of the student. The data type of this attribute is numeric. Values vary from 1 to 4. 1 – less than 2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 – more than 10 hours.
* failures
* This attribute refers to several past class failures of the student. The data type of this attribute is numeric. If 0 to 3 failures, the number of failures is taken, else 4 is taken.
* schoolsup
* This attribute refers to the extra educational support of the student. The data type of this attribute is binary. The binary value is either yes or no.
* famsup
* This attribute refers to the family educational support of the student. The data type of this attribute is binary. The binary value is either yes or no.
* paid
* This attribute refers to extra paid classes within the course subject of the student. The data type of this attribute is binary. The binary value is either yes or no.
* activities
* This attribute refers to the extra-curricular activities of the student. The data type of this attribute is binary. The binary value is either yes or no.
* nursery
* This attribute refers to the student attending nursery school. The data type of this attribute is binary. The binary value is either yes or no.
* higher
* This attribute refers to whether the student wants to take higher education of the student. The data type of this attribute is binary. The binary value is either yes or no.
* internet
* This attribute refers to internet access at the home for the student. The data type of this attribute is binary. The binary value is either yes or no.
* romantic
* This attribute refers to the romantic relationships of the student. The data type of this attribute is binary. The binary values are either yes or no.
* famrel
* This attribute refers to the family relationship status of the student. The data type for this attribute is numeric (integer) and its values can vary from 1 to 5, 1 being very bad relationship status and 5 being excellent relationship status.
* freetime
* This attribute refers to the free time after school of the student. The data type of this attribute is numeric (integer), and its values can vary from 1 to 5, 1 being very low to 5 being very high.
* goout
* This attribute refers to going out with friends of the student. The data type of this attribute is numeric (integer), and its values can vary from 1 to 5, 1 being very low to 5 being very high.
* Dalc
* This attribute refers to the workday alcohol consumption of the student. The data type of this attribute is numeric (integer), and its values can vary from 1 to 5, 1 being very low to 5 being very high.
* Walc
* This attribute refers to the weekend alcohol consumption of the student. The data type of this attribute is numeric (integer), and its values can vary from 1 to 5, 1 being very low to 5 being very high.
* health
* This attribute refers to the current health status of the student. The data type of this attribute is numeric (integer), and its values can vary from 1 to 5, 1 being very low to 5 being very high.
* absences
* This attribute refers to the number of school absences of the student. The data type of this attribute is numeric (integer), and its values can vary from 0 to 93.
* G1
* This attribute refers to the first-term grade of the student. The data type of this attribute is numeric (integer), and its values can vary from 0 to 20.
* G2
* This attribute refers to the second-term grade of the student. The data type of this attribute is numeric (integer), and its values can vary from 0 to 20.
* G3
* The target attribute refers to the final grade and it has a strong correlation with the attributes G2 and G1. The data type of this attribute is numeric(integer), and its values can vary from 0 to 20.

We have analyzed the data distribution of the attributes of the dataset and plotted the distributions in pie charts, bar charts, and line graphs. Some of these graphs are mentioned below.

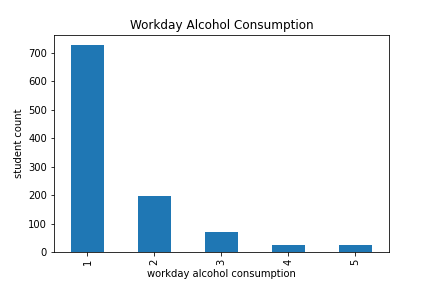
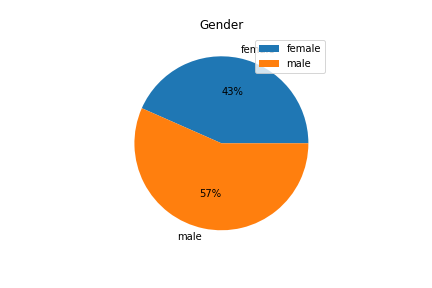


Figure 1: Pie chart showing the data distribution based on gender of the students

Figure 2: Bar chart showing the workday alcohol consumption of the students

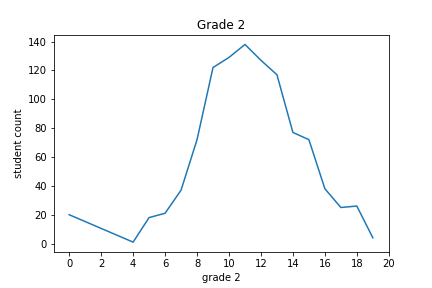
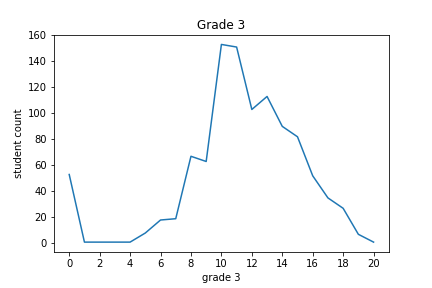


Figure 3: Line graph showing the data distribution for the second term grades

Figure 4: Line graph showing the data distribution for the final examination grades

The target variable of the student performance data set is G3. It refers to the final examination grades of the students. We have analyzed the correlation between the target variable and other attributes of the dataset by drawing scatter plots.

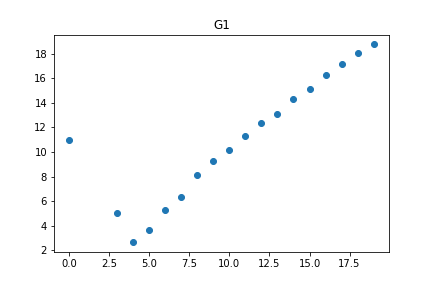
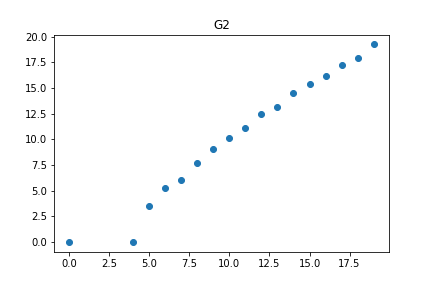


Figure 6: Correlation between G2 and G3

Figure 5: Correlation between G1 and G3

The above scatter plots demonstrate that there is a strong correlation between G1/G3 and G2/G3. Further, the G3 value is proportional to the G1 and G2 values.

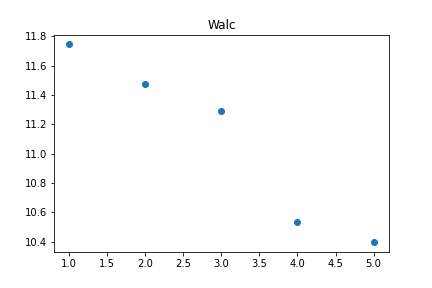
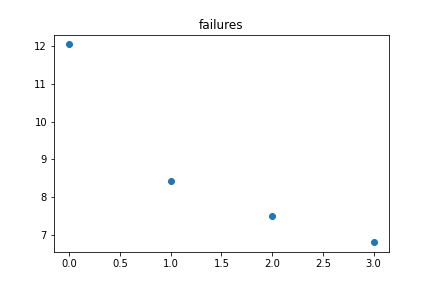
 Th

Figure 8: Correlation between failures and G3

Figure 7: Correlation between Walc and G3

The above plots denote the relationship between the target variable G3 and other attributes such as Walc (workday alcohol consumption) and failures. It shows that G3 is inversely proportional to the failures and workday alcohol consumption.

Methodology

Data Cleaning and Preprocessing

Real-world data sets contain several problems such as missing or null values, data inconsistency, incompleteness, and outliers of the dataset. Therefore, data preprocessing is a mandatory step before feeding the dataset into a machine learning model. Data preprocessing includes transforming row data into the machine-understandable and efficient format. Since the student performance dataset also contains some of these issues, we had to do preprocessing tasks before using the dataset for model training purposes.

As the initial step of preprocessing, we imported the two student performance datasets (Mathematics and Portuguese language) as Pandas DataFrames. After that, the two Pandas DataFrames were merged into one.

df\_mat = pd.read\_csv('../dataset/original/student-mat.csv', sep=';')

df\_por = pd.read\_csv('../dataset/original/student-por.csv', sep=';')

merged\_df = pd.concat([df\_mat, df\_por])

Then the dataset was analyzed to check whether there are any missing or null values existed within the dataset. If any null values existed within the dataset, there are usually two ways to deal with them. The first method is to delete the particular rows which contain multiple null values. The second method is to replace the missing values with mean, median, or most frequent values. Although our analysis proved that the student performance dataset did not contain any missing or null values, we implemented the null value removal step programmatically.

merged\_df = merged\_df[merged\_df.notna().all(axis=1)]

Furthermore, the dataset was checked to identify duplicate rows. Although our analysis proved that the student performance dataset did not contain any duplicate rows, we implemented the duplicate row removal step programmatically.

merged\_df = merged\_df.drop\_duplicates()  
merged\_df.duplicated().sum()

To address the selected problem, we decided to categorize the students into multiple classes based on their final examination grades (G3) as shown in the below table.

|  |  |  |
| --- | --- | --- |
| **Class** | **G3 Range** | **Status** |
| Weak | 0 <= G3 <= 7 | 0 |
| Intermediate | 8 <= G3 <= 14 | 1 |
| Good | 15 <= G3 <= 20 | 2 |

The table indicates that status describes the students’ performance level based on their G3 value. Therefore, this problem becomes a classification problem rather than a regression problem since we will be predicting the class to which the student belongs.

merged\_df['status'] = merged\_df.apply(*lambda* x: 0 *if* x['G3'] <= 7 *else* (1 *if* x['G3'] <= 14 *else* 2), axis=1)

The status variable contains numerical data (0, 1, 2) instead of string or categorical data (Weak, Intermediate, Good) to make the implementation of Machine Learning algorithms more efficient. Afterward, we dropped the G3 column since now the status column represents the target variable in the classification problem and G3 is just a duplicate representation of the status.

merged\_df.drop(['G3'], axis='columns', inplace=*True*)

As the next preprocessing step, we encoded the categorical data in the dataset. It involves converting categorical data into numerical data. This preprocessing step is mandatory in Machine Learning since many Machine Learning algorithms require input and output data to be numerical values for more efficient performance. Label Encoding and One Hot Encoding are the methods that we used to encode categorical data.

Scikit-learn LabelEncoder converts every categorical value in a column to a number.

*for* column *in* string\_columns:  
 X.loc[:, column] = label\_encoder.fit\_transform(X.loc[:, column])

Afterward, we applied Scikit-learn OneHotEncoder for the dataset. The specialty of this method is that it solves the hierarchical issue of label encoding. So, if a column has three categories, three columns will be created, and likewise for any number of categories.

X['guardian\_father'] = one\_hot\_encoder.fit\_transform(X[['guardian']])[:, 0]  
X['guardian\_mother'] = one\_hot\_encoder.fit\_transform(X[['guardian']])[:, 1]  
X['guardian\_other'] = one\_hot\_encoder.fit\_transform(X[['guardian']])[:, 2]

We applied One Hot Encoding for Mjob, Fjob, reason, and guardian attributes. Then we dropped the initial four attributes from the DataFrame.

Then we plotted the heatmap of the correlation matrix. It demonstrates the correlation coefficient between attributes of the dataset. Each cell in the heatmap shows how two variables are related.

sns.heatmap(X[X.columns].corr().abs(), annot=*True*)

This is an important step for the preprocessing of the dataset because highly correlated attributes should not be included in the training dataset. We can easily identify highly correlated attributes through the correlation matrix and remove one from each correlated pair from the DataFrame.

upper\_tri = correlation\_matrix.where(np.triu(np.ones(correlation\_matrix.shape), k=1).astype(np.bool))

to\_drop\_pairs = []  
  
*for* i *in range*(*len*(upper\_tri.columns)):  
 *if any*(upper\_tri[upper\_tri.columns[i]] > 0.6):  
 to\_drop\_pairs.append([upper\_tri.columns[i], upper\_tri.columns[i - 1]])

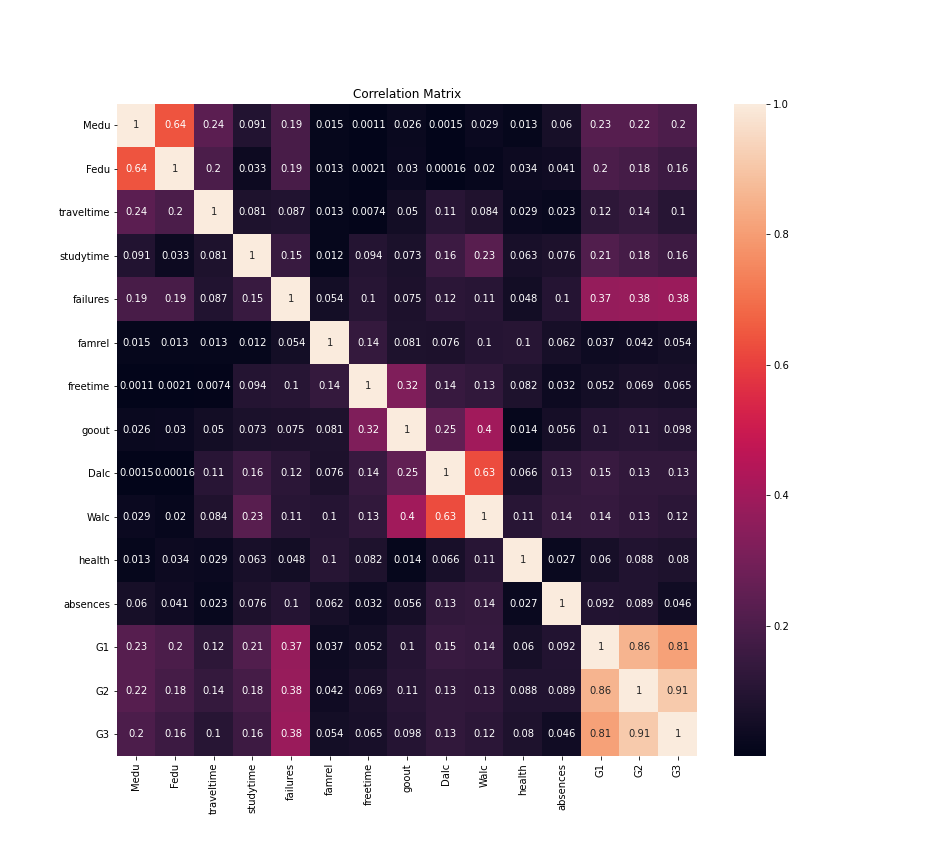


Figure 9: Heatmap showing the correlation among variables

As the next preprocessing step, we analyzed the feature importance of the dataset and performed dimensionality reduction. Feature importance identifies the relatively important features for the prediction out of all existing features of the dataset. XGBoost Feature Importance was used to analyze the importance of features. Identifying the important features of the dataset, feature selection, and dimensionality reduction help to improve the accuracy and effectiveness of the model.

model = XGBClassifier()  
model.fit(X, y)  
  
importances = model.feature\_importances\_  
  
importance\_list = []  
  
*for* i, v *in enumerate*(importances):  
 importance\_list.append([i, X.columns[i], v])  
  
importance\_list = *sorted*(importance\_list, reverse=*True*, key=*lambda* x: x[2])

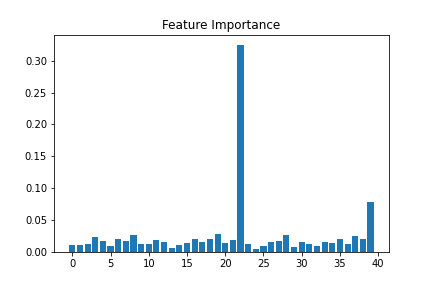


Figure 10: Bar graph showing the feature importance

According to the results given by XGBoost Feature Importance we decided to select the top 12 most important features to train the model. The top 12 important features, their index, and their feature importance values are given below.

[[22, 'G2', 0.32473215],

[39, 'guardian\_other', 0.07848206],

[19, 'Dalc', 0.028584061],

[28, 'father\_job\_at\_home', 0.026292209],

[8, 'schoolsup', 0.025903812],

[37, 'guardian\_father', 0.025244838],

[3, 'Pstatus', 0.02388544],

[16, 'famrel', 0.021134308],

[35, 'reason\_other', 0.021047307],

[38, 'guardian\_mother', 0.020905472],

[6, 'studytime', 0.020557376],

[18, 'goout', 0.020325337],

After dimensionality reduction, the dataset was split into train and test sets. The training set was 80% of the entire dataset while the test set was 20% of the entire dataset.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

|  |  |
| --- | --- |
| Total dataset rows | 1044 |
| Train dataset rows | 835 |
| Test dataset rows | 209 |

Machine Learning Algorithms

We trained four different Machine Learning algorithms to predict student performance. They are the K-Nearest Neighbors (KNN) algorithm, Decision Tree algorithm, Random Forest algorithm, and Naive Bayes algorithm. After evaluating each trained model, the results show that KNN algorithm provides the highest accuracy in predicting the correct student performance status.

Justification for Choosing the KNN Algorithm

The student performance prediction problem can be addressed through supervised learning techniques. In supervised learning, a data set, with a set of features and labels, is fed to the learning algorithm. When the algorithm is fully trained, it will correctly identify the relationships between features and their labels. Therefore, it will be able to accurately predict the label for a previously unseen set of features. Since the student performance, the dataset contains labeled data a supervised learning algorithm should be used for this problem. Therefore, supervised learning algorithms such as KNN algorithm, the Decision Tree algorithm, the Random Forest algorithm, and the Naive Bayes algorithm can be used.

There are two main types of supervised learning problems called Classification problems and Regression problems. Regression is used when the supervised machine learning problem is predicting a numerical label. Classification is used when the supervised machine learning problem is predicting a class label. Since in the preprocessing stage, we decided to insert a column called status with three classes called Weak (0), Intermediate (1), and Good (2) instead of the original numeric target variable (G3) this problem should be addressed through a classification algorithm. There are two types of classification algorithms called binary classification and multi-class classification. Binary classification categorizes data into one of two categories such as True (1) or False (0). Multiclass classification categorizes data into one of the multiple number of classes. Since the student performance problem’s target variable contains three classes (Weak, Intermediate, Good), multi-class classification is applicable for this scenario. Even though KNN algorithm, Decision Tree algorithm, Random Forest algorithm, and Naive Bayes algorithm can be used to solve multi-class classification problems, research shows that KNN algorithm is more effective.

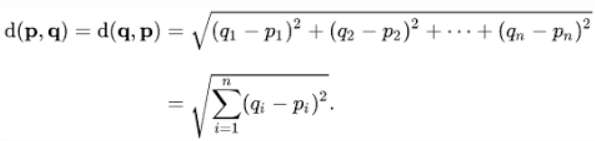
Further, KNN is a very simple, easy-to-understand algorithm which is also easier to implement. Only a few parameters are there to tune. It has very less calculation time as well. There are no requirements to make additional assumptions. It is also known to provide higher accuracy when compared to other supervised learning multi-class classification algorithms.

Introduction and Background of KNN Algorithm

KNN is a simple machine learning algorithm based on supervised learning, which can be used for both regression and classification problems. This algorithm was developed by Evelyn Fix and Joseph Hodges in the year of 1951 and later it was improved by Thomas Cover. KNN is a lazy learner algorithm because, instead of learning from the training dataset, it memorizes the data set and performs the action at the time of classification.

The quality of the predictions made by the algorithm depends on the distance measures. KNN is a distance-based classifier, it functions by finding the distance between examples in the data and the query, closer the two points are to each other the greater their similarities are in behavior.

There are four methods to measure the distance, which are Euclidean distance, Hamming distance, Manhattan distance, and Minkowski distance. The Euclidean distance formula is as followed,



In KNN, the “K” value represents the number of nearest neighbors. A very low value for “K” may lead to the effects of outliers in the model and a larger value makes it computationally expensive. Therefore, the most preferred value for “K” is considered as value 5.

The functionality of KNN can be demonstrated in three steps,

1. Calculate distance
2. Find closest neighbors
3. Vote for labels

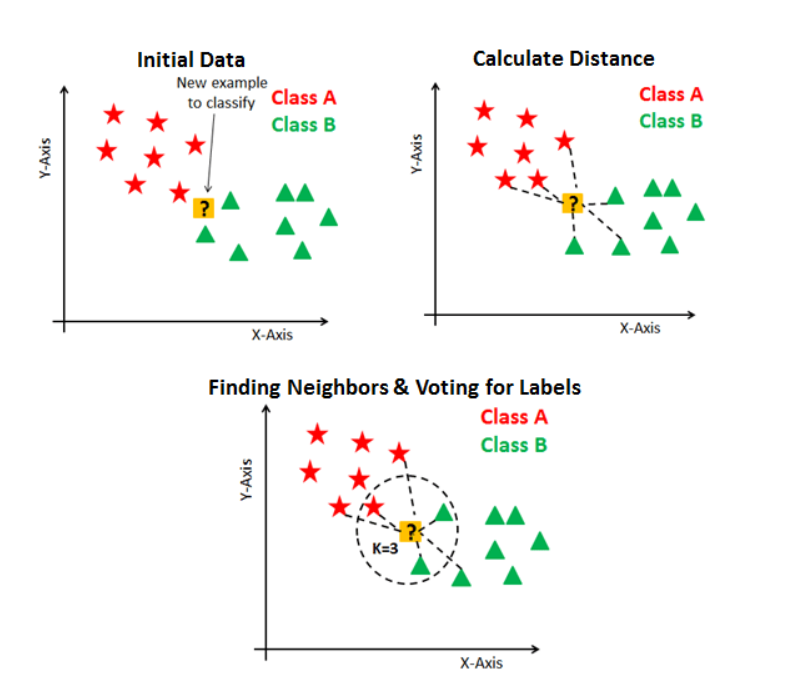


Figure 1: Functionality of KNN

The mathematical process behind the KNN algorithm is as follows. First, the algorithm finds the “K” closest point to the label that needs to be predicted. After that, the algorithm classifies the points by the majority vote of its “K” neighbors. Then each neighbor votes for their class and the class which contains the majority votes is taken as the prediction. The algorithm finds the closest similar points by measuring the distance between points using the above-mentioned distance measures.

KNN algorithm provides several benefits. It is a very simple algorithm that is also easier to implement. It is very easy to understand because only a few parameters are there to tune. It has very less calculation time as well. There are no requirements to make additional assumptions. It is also known to provide higher accuracy when compared to other supervised learning multi-class classification algorithms. The developer has the freedom to choose a flexible distance criterion when building a KNN model such as Euclidean distance, Hamming distance, Manhattan distance, and Minkowski distance.

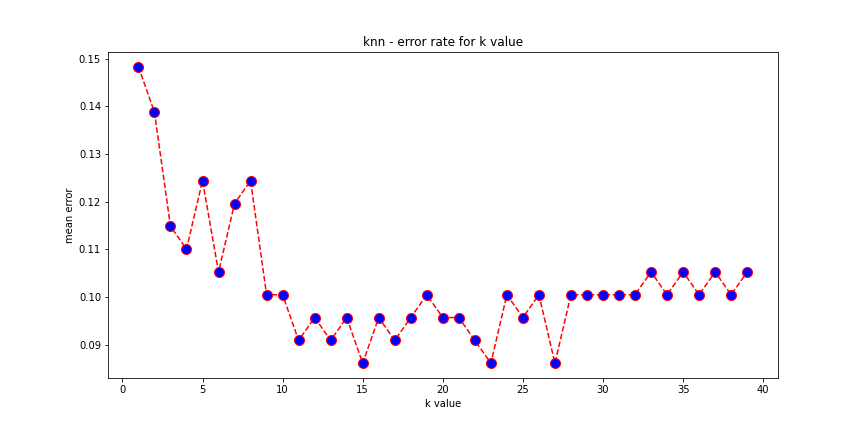
The major limitation of the KNN algorithm is the inability to address any missing or null values. To overcome this limitation, we can perform data preprocessing techniques such as deleting the rows which contain one or more null values or filling the missing values with the average value of the features across the entire dataset before defining the model.

As real-world examples for multiclass classification problems using KNN are handwritten digit recognition, student performance prediction, video recognition, image recognition, and cancer stage identification.

Implementation

Since there is not any specific method to determine the best value for “K”, we decided to measure the “K” value which provides the minimum error rate in the range from 1 to 40. Then we plotted the line graph to visualize the error rate against the “K” value.

error = []  
  
*for* i *in range*(1, 40):  
 knn = KNeighborsClassifier(n\_neighbors=i)  
 knn.fit(X\_train, y\_train)  
 pred\_i = knn.predict(X\_test)  
 error.append(np.mean(pred\_i != y\_test))  
  
best\_k = error.index(*min*(error)) + 1



Figure

Figure 12: Line graph to visualize the error rate against “K” value

Based on the best “K” value we trained the KNN model.

knn\_model = KNeighborsClassifier(n\_neighbors=best\_k)

knn\_model.fit(X\_train, y\_train)

y\_pred\_knn = knn\_model.predict(X\_test)

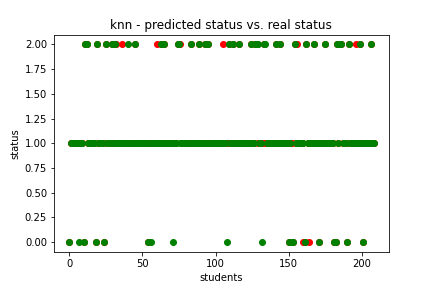


Figure 13: Scatter plot showing the predicted status vs. real status for KNN algorithm

The above scatter plot visualizes the predicted target data points in red and test target data points in green for the KNN algorithm. Red dots represent inaccurate predictions.

Results and Discussion

Model Evaluation

We evaluated the four models that we trained in different evaluation metrics such as accuracy score, precision score, recall score, f1-score, Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Confusion matrix.

Accuracy refers to the ratio of the number of correct predictions to the total number of input samples.



*print*(accuracy\_score(y\_test, y\_pred))

[('KNN', 0.9138755980861244),

('RF', 0.8995215311004785),

('DT', 0.8851674641148325),

('NB', 0.8229665071770335)]

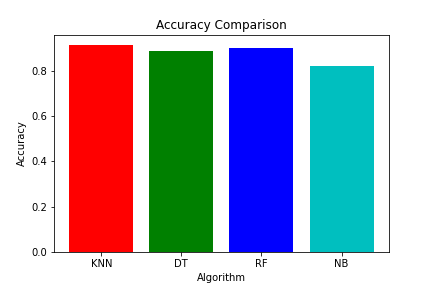


Figure 14: Accuracy comparison

The accuracies of different supervised learning algorithms are demonstrated on the above bar chart. According to the bar chart, KNN algorithm had gained the highest accuracy.

True Positives: The cases in which the model predicted true, and the actual output was also true.

True Negatives: The cases in which the model predicted false, and the actual output was also false.

False Positives: The cases in which the model predicted true, and the actual output was false.

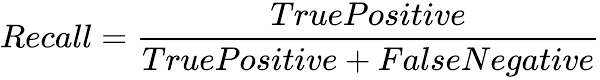
False Negatives: The cases in which the model predicted false, and the actual output was true.

Precision denotes the ratio of the number of correct positive results and the number of positive results.

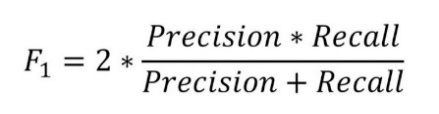


precision[y\_pred] = precision\_score(y\_test, y\_pred\_dict[y\_pred], average='weighted')

Recall denotes the ratio of the number of correct positive results and the number of all samples.



recall[y\_pred] = recall\_score(y\_test, y\_pred\_dict[y\_pred], average='weighted')

F1-score refers to the weighted average of the precision score and recall score.

f1[y\_pred] = f1\_score(y\_test, y\_pred\_dict[y\_pred], average='weighted')

print(classification\_report(y\_test, y\_pred))

precision recall f1-score support

0 0.87 0.68 0.76 19

1 0.94 0.94 0.94 153

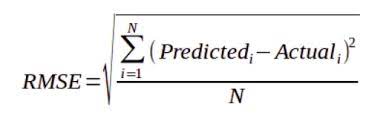
2 0.83 0.92 0.87 37

accuracy 0.91 209

macro avg 0.88 0.85 0.86 209

weighted avg 0.91 0.91 0.91 209

Root Mean Square Error (RMSE) is the square root of the average of the squared differences between the predicted and the actual value of the variable.



*print*(mean\_squared\_error(y\_test, y\_pred, squared=*False*))

*print*(mean\_absolute\_error(y\_test, y\_pred))

The confusion matrix is a summarized tableused to assess the performance of a classification model. The number of correct and incorrect predictions are summarized with their count according to each class.

Figure 15: Confusion matrix

*print*(confusion\_matrix(y\_test, y\_pred))

cm\_plot = plot\_confusion\_matrix(model, X\_test, y\_test)

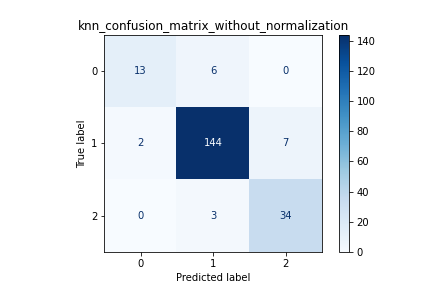
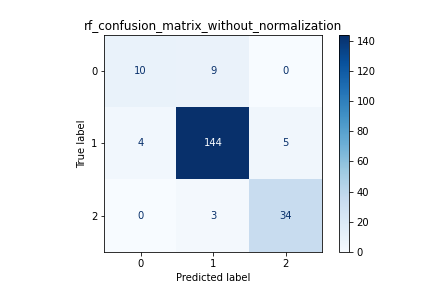


Figure 17: Confusion matrix for Random Forest algorithm

Figure 16: Confusion matrix for KNN algorithm

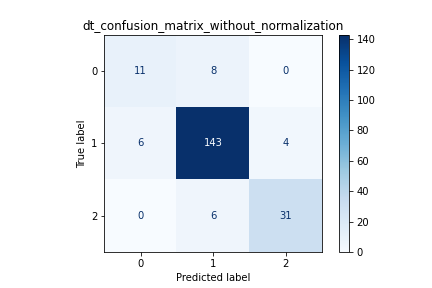
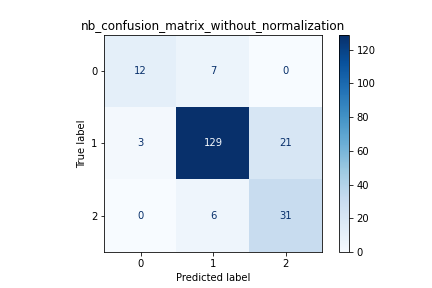


Figure 19: Confusion matrix for Naïve Bayes algorithm

Figure 18: Confusion matrix for Decision Tree algorithm

A summary of evaluation metric results is presented below in the table and graph.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **K-Nearest Neighbors** | **Random Forest** | **Decision Tree** | **Naïve Bayes** |
| **Accuracy Score** | 0.9139 | 0.8995 | 0.8852 | 0.8230 |
| **Precision Score** | 0.9146 | 0.8950 | 0.8824 | 0.8433 |
| **Recall Score** | 0.9139 | 0.8995 | 0.8852 | 0.8230 |
| **F1 Score** | 0.9129 | 0.8958 | 0.8834 | 0.8277 |
| **RMSE** | 0.2935 | 0.3170 | 0.3389 | 0.4208 |
| **MAE** | 0.0861 | 0.1005 | 0.1148 | 0.1770 |

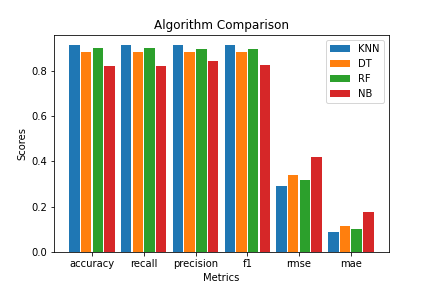


Figure 20: Algorithm comparison

These evaluation matrix results clearly show that KNN algorithm acquires the highest accuracy and efficiency when predicting student performances. Therefore, the most appropriate supervised learning algorithm for this problem is KNN algorithm.

Accuracy Improvement Techniques, Limitations, and Future Work

Even though, KNN algorithm showed high accuracy of 0.9139 when predicting the student performance, the accuracy can be further improved through different techniques. The dataset can be enlarged so that the model can do more accurate predictions rather than relying on assumptions and weaker correlations between attributes. Furthermore, in the preprocessing stage, the outliers in the dataset can be detected and removed. Additionally, more feature engineering and feature selection techniques such as domain knowledge, visualization, and statistical parameters (PCA – Principal Component Analysis) based methods can be applied to find out the best subset of features that explain the relationships of independent features with target variable better.

In this solution, we tested four supervised machine learning models namely K-Nearest Neighbors (KNN), Random Forest, Decision Tree, and Naïve Bayes. The KNN model provided the best results according to our observations. But other supervised machine learning algorithms for multiclass classification such as Support Vector Machine (SVM) can be tested and evaluated to find out whether they can acquire better results. Furthermore, more investigation can be carried out to find out the optimum values for hyper-parameters for different learning algorithms through algorithm tuning. For example, maximum number of features, number of trees, random state, and OOB score parameters of Random Forest algorithm and number of neighbors in KNN algorithm etc.

The above-implemented student performance prediction process can be further enhanced to gain better results. The dataset can be further enlarged with the data of different schools and different age groups. Additionally, the data set can be expanded among different countries outside of Portugal as well. For example, student performances in Sri Lankan examinations such as GCE Ordinary Level (O/L) and GCE Advanced level (A/L) can be added to the dataset and the performances for Sri Lankan students for the above examinations can be predicted.

Furthermore, more research can be done on sociological studies to identify how different features affect student performance. Different feature selection methods can be used to gain better results. For example, automatic feature selection methods such as wrapping, and filtering can experiment in the future. Finally, for student performance prediction, non-linear methods such as Neural Networks can be evaluated to acquire better and more accurate results.

References

[1] P. Cortez and A. Silva. Using Data Mining to Predict Secondary School Student Performance. In A. Brito and J. Teixeira Eds., Proceedings of 5th Future Business Technology Conference (FUBUTEC 2008) pp. 5-12, Porto, Portugal, EUROSIS, ISBN 978-9077381-39-7, April 2008.

[2] F. Ünal, “Data mining for student performance prediction in education,” in Data Mining - Methods, Applications, and Systems, D. Birant, Ed. London, England: IntechOpen,2021. 10.5772/intechopen.91449, 2020.

[3] Walia, Nidhi & Kumar, Mukesh. “Student's Academic Performance Prediction in Academic using Data Mining Techniques,” 1st International Conference on Intelligent Communication and Computational Research (ICICCR), 2020.

[4] F. J. Kaunang and R. Rotikan, “Students' Academic Performance Prediction using Data Mining,” 2018 Third International Conference on Informatics and Computing (ICIC), 2018, pp. 1-5, doi: 10.1109/IAC.2018.8780547.

[5] E. H. Yossy, Y. Heryadi and Lukas, “Comparison of Data Mining Classification Algorithms for Student Performance,” 2019 IEEE International Conference on Engineering, Technology and Education (TALE), 2019, pp. 1-4, doi: 10.1109/TALE48000.2019.9225887.

[6] Rahul and R. Katarya, “Impact of Supervised Classification Techniques for the Prediction of Student's Performance,” 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics, and Cloud) (I-SMAC), 2020, pp. 537-543, doi: 10.1109/I-SMAC49090.2020.9243360.

[7] Liu Ruochena, Mei Wenjuanb, and Liu Junc, “Research on Predicting Students’ Performance Based on Machine Learning,” 2018 International Conference on Big Data and Artificial Intelligence (ICBDA), 2018.

[8] Madnaik, Sandeep Subhash, “Predicting Students’ Performance by Learning Analytics,”. Master's Projects. 941. DOI: https://doi.org/10.31979/etd.6jjb-ua9w, 2020.

[9] Imran, Muhammad & Latif, Shahzad & Mehmood, Danish & Shah, Muhammad, “Student Academic Performance Prediction using Supervised Learning Techniques,” International Journal of Emerging Technologies in Learning (iJET). 14. 92. 10.3991/ijet.v14i14.10310. (2019)

[10] V.Ramesh, P.Parkavi, K.Ramar, “Predicting Student Performance: A Statistical and Data Mining Approach,” International Journal of Computer Applications (0975 – 8887) Volume 63– No.8, February 2013.

[11] Sonali Kashyap, Shikhar Bajaj, Jabanjalin Hida, “S Prediction of Student Academics Performance Based on their Previous Performance and Family Support,” School of Computing Science and Engineering, Vellore Institute of Technology Vellore, India, DOI:10.5958/0974-360X.2017.00362.6, 2017.

Appendixes

#!/usr/bin/env python  
# coding: utf-8  
  
# In[1]:  
  
  
# importing pandas library to perform data manipulation and analysis  
import pandas as pd  
  
pd.options.display.max\_columns = None  
  
  
# In[2]:  
  
  
# importing the student-mat.csv dataset to a pandas dataframe  
# dataset includes the student performances for mathematics subject  
df\_mat = pd.read\_csv(**'../dataset/original/student-mat.csv'**, sep=**';'**)  
df\_mat.insert(0, **'Subject'**, 0)  
df\_mat  
  
  
# In[3]:  
  
  
# importing the student-por.csv dataset to a pandas dataframe  
# dataset includes the student performances for portuguese subject  
df\_por = pd.read\_csv(**'../dataset/original/student-por.csv'**, sep=**';'**)  
df\_por.insert(0, **'Subject'**, 1)  
df\_por  
  
  
# In[4]:  
  
  
# merging the two dataframes into one  
merged\_df = pd.concat([df\_mat, df\_por])  
merged\_df = merged\_df.reset\_index(drop=True)  
merged\_df  
  
  
# In[5]:  
  
  
# dropping irrelevant attribute columns from the dataframe  
merged\_df.drop([**'Subject'**, **'school'**, **'age'**], axis=**'columns'**, inplace=True)  
merged\_df  
  
  
# In[6]:  
  
  
# adding a new column called status to the dataframe based on the G3 value  
# categorizing the students into three classes based on their final examination grades  
merged\_df[**'status'**] = merged\_df.apply(lambda x: 0 if x[**'G3'**] <= 7 else (1 if x[**'G3'**] <= 14 else 2), axis=1)  
merged\_df  
  
  
# In[7]:  
  
  
# removing the missing or null values from the dataframe if exist  
merged\_df = merged\_df[merged\_df.notna().all(axis=1)]  
  
# checking for missing or null values in the dataframe  
merged\_df\_null = merged\_df[merged\_df.isnull().any(axis=1)]  
merged\_df\_null  
  
  
# In[8]:  
  
  
# importing pyplot from matplotlib library to create interactive visualizations  
import matplotlib.pyplot as plt  
  
# importing seaborn library which is built on top of matplotlib to create statistical graphics  
import seaborn as sns  
  
# plotting the heatmap for missing or null values in the dataframe  
fig = plt.figure(figsize=(9, 8))  
sns.heatmap(merged\_df.isnull(), yticklabels=False, cbar=False, cmap=**'viridis'**)  
plt.title(**'Null Values Detection Heat Map'**)  
fig.savefig(**'preprocessing\_plots/null\_detection\_heat\_map.png'**, facecolor=**'white'**)  
plt.show()  
  
  
# In[9]:  
  
  
# removing the duplicate rows from the dataframe if exist  
merged\_df = merged\_df.drop\_duplicates()  
  
# checking the number of duplicate rows exist in the dataframe  
merged\_df.duplicated().sum()  
  
  
# In[10]:  
  
  
merged\_df  
  
  
# In[11]:  
  
  
# displaying the first 5 rows of the dataframe  
merged\_df.head()  
  
  
# In[12]:  
  
  
# displaying the last 5 rows of the dataframe  
merged\_df.tail()  
  
  
# In[13]:  
  
  
# displaying the dimensionality of the dataframe  
merged\_df.shape  
  
  
# In[14]:  
  
  
# printing a concise summary of the dataframe  
# information such as index, data type, columns, non-null values, and memory usage  
merged\_df.info()  
  
  
# In[15]:  
  
  
# copying the dataset without the status attribute  
df\_without\_status = merged\_df.loc[:, merged\_df.columns != **'status'**]  
df\_without\_status  
  
  
# In[16]:  
  
  
# generating descriptive statistics of the dataframe  
df\_without\_status.describe().round(2)  
  
  
# In[17]:  
  
  
# displaying the unbiased variance of the numeric columns of the dataframe  
variance\_vector = df\_without\_status.var()  
variance\_vector  
  
  
# In[18]:  
  
  
# computing and displaying pair-wise correlation of columns of the dataframe  
correlation\_matrix = df\_without\_status.corr().abs()  
correlation\_matrix  
  
  
# In[19]:  
  
  
# sorting the attribute correlation values with G3 in descending order  
correlation\_matrix[**'G3'**].sort\_values(ascending=False)  
  
  
# In[20]:  
  
  
# importing numpy library to perform fast mathematical operations over arrays and matrices  
import numpy as np  
  
upper\_tri = correlation\_matrix.where(np.triu(np.ones(correlation\_matrix.shape), k=1).astype(np.bool))  
upper\_tri  
  
  
# In[21]:  
  
  
# identifying one attribute out of each highly correlated column pair  
to\_drop = [column for column in upper\_tri.columns if any(upper\_tri[column] > 0.6)]  
to\_drop  
  
  
# In[22]:  
  
  
# identifying highly correlated column pairs  
  
to\_drop\_pairs = []  
  
for i in range(len(upper\_tri.columns)):  
 if any(upper\_tri[upper\_tri.columns[i]] > 0.6):  
 to\_drop\_pairs.append([upper\_tri.columns[i], upper\_tri.columns[i - 1]])  
  
to\_drop\_pairs  
  
  
# In[23]:  
  
  
# plotting the heatmap for correlation matrix  
fig = plt.figure(figsize=(13, 12))  
sns.heatmap(df\_without\_status[df\_without\_status.columns].corr().abs(), annot=True)  
plt.title(**'Correlation Matrix'**)  
fig.savefig(**'preprocessing\_plots/correlation\_matrix.png'**, facecolor=**'white'**)  
plt.show()  
  
  
# In[24]:  
  
  
from pandas.api import types  
  
# identifying the columns with numeric values  
numeric\_columns = [i for i in merged\_df.columns if types.is\_numeric\_dtype(merged\_df[i])]  
numeric\_columns  
  
  
# In[25]:  
  
  
# identifying the columns with string values  
string\_columns = [i for i in merged\_df.columns if types.is\_string\_dtype(merged\_df[i])]  
string\_columns  
  
  
# In[26]:  
  
  
# dropping G3 column  
# status column represents the target variable now  
# G3 is just a duplicate representation of the status  
merged\_df.drop([**'G3'**], axis=**'columns'**, inplace=True)  
merged\_df  
  
  
# In[27]:  
  
  
# assigning all attributes except status (features) to X  
X = merged\_df.loc[:, merged\_df.columns != **'status'**]  
X  
  
  
# In[28]:  
  
  
# assigning status (target) to y  
y = merged\_df.loc[:, **'status'**]  
y  
  
  
# In[29]:  
  
  
# importing LabelEncoder from scikit-learn library  
# converting every categorical value in a column to a number  
from sklearn.preprocessing import LabelEncoder  
  
label\_encoder = LabelEncoder()  
  
for column in string\_columns:  
 X.loc[:, column] = label\_encoder.fit\_transform(X.loc[:, column])  
  
X  
  
  
# In[30]:  
  
  
# importing OneHotEncoder from scikit-learn library  
# solving the hierarchical issue of label encoding  
from sklearn.preprocessing import OneHotEncoder  
  
one\_hot\_encoder = OneHotEncoder(sparse=False)  
  
# since Mjob contains 5 categories, 5 columns will be created  
X[**'mother\_job\_at\_home'**] = one\_hot\_encoder.fit\_transform(X[[**'Mjob'**]])[:, 0]  
X[**'mother\_job\_health'**] = one\_hot\_encoder.fit\_transform(X[[**'Mjob'**]])[:, 1]  
X[**'mother\_job\_other'**] = one\_hot\_encoder.fit\_transform(X[[**'Mjob'**]])[:, 2]  
X[**'mother\_job\_services'**] = one\_hot\_encoder.fit\_transform(X[[**'Mjob'**]])[:, 3]  
X[**'mother\_job\_teacher'**] = one\_hot\_encoder.fit\_transform(X[[**'Mjob'**]])[:, 4]  
  
# since Fjob contains 5 categories, 5 columns will be created  
X[**'father\_job\_at\_home'**] = one\_hot\_encoder.fit\_transform(X[[**'Fjob'**]])[:, 0]  
X[**'father\_job\_health'**] = one\_hot\_encoder.fit\_transform(X[[**'Fjob'**]])[:, 1]  
X[**'father\_job\_other'**] = one\_hot\_encoder.fit\_transform(X[[**'Fjob'**]])[:, 2]  
X[**'father\_job\_services'**] = one\_hot\_encoder.fit\_transform(X[[**'Fjob'**]])[:, 3]  
X[**'father\_job\_teacher'**] = one\_hot\_encoder.fit\_transform(X[[**'Fjob'**]])[:, 4]  
  
# since reason contains 4 categories, 4 columns will be created  
X[**'reason\_course'**] = one\_hot\_encoder.fit\_transform(X[[**'reason'**]])[:, 0]  
X[**'reason\_home'**] = one\_hot\_encoder.fit\_transform(X[[**'reason'**]])[:, 1]  
X[**'reason\_other'**] = one\_hot\_encoder.fit\_transform(X[[**'reason'**]])[:, 2]  
X[**'reason\_reputation'**] = one\_hot\_encoder.fit\_transform(X[[**'reason'**]])[:, 3]  
  
# since guardian contains 3 categories, 3 columns will be created  
X[**'guardian\_father'**] = one\_hot\_encoder.fit\_transform(X[[**'guardian'**]])[:, 0]  
X[**'guardian\_mother'**] = one\_hot\_encoder.fit\_transform(X[[**'guardian'**]])[:, 1]  
X[**'guardian\_other'**] = one\_hot\_encoder.fit\_transform(X[[**'guardian'**]])[:, 2]  
  
X  
  
  
# In[31]:  
  
  
# dropping the initial four attributes from the dataframe  
X.drop([**'Mjob'**, **'Fjob'**, **'reason'**, **'guardian'**], axis=**'columns'**, inplace=True)  
X  
  
  
# In[32]:  
  
  
# displaying the unbiased variance of the numeric columns of the dataframe  
variance\_vector = X.var()  
variance\_vector  
  
  
# In[33]:  
  
  
# computing and displaying pair-wise correlation of columns of the dataframe  
correlation\_matrix = X.corr().abs()  
correlation\_matrix  
  
  
# In[34]:  
  
  
# plotting the heatmap for correlation matrix  
fig = plt.figure(figsize=(40, 40))  
sns.heatmap(X[X.columns].corr().abs(), annot=True)  
plt.title(**'Correlation Matrix for Features'**)  
fig.savefig(**'preprocessing\_plots/correlation\_matrix\_for\_features.png'**, facecolor=**'white'**)  
plt.show()  
  
  
# In[35]:  
  
  
upper\_tri = correlation\_matrix.where(np.triu(np.ones(correlation\_matrix.shape), k=1).astype(np.bool))  
upper\_tri  
  
  
# In[36]:  
  
  
# identifying one attribute out of each highly correlated column pair  
to\_drop = [column for column in upper\_tri.columns if any(upper\_tri[column] > 0.6)]  
to\_drop  
  
  
# In[37]:  
  
  
# identifying highly correlated column pairs  
  
to\_drop\_pairs = []  
  
for i in range(len(upper\_tri.columns)):  
 if any(upper\_tri[upper\_tri.columns[i]] > 0.6):  
 to\_drop\_pairs.append([upper\_tri.columns[i], upper\_tri.columns[i - 1]])  
  
to\_drop\_pairs  
  
  
# In[38]:  
  
  
# dropping one attribute out of each highly correlated column pair  
X.drop([**'Fedu'**, **'Walc'**, **'G1'**], axis=**'columns'**, inplace=True)  
X  
  
  
# In[39]:  
  
  
# importing XGBClassifier from xgboost library  
from xgboost import XGBClassifier  
  
model = XGBClassifier()  
model.fit(X, y)  
  
# detecting feature importance of the training attributes  
importances = model.feature\_importances\_  
  
importance\_list = []  
  
for i, v in enumerate(importances):  
 importance\_list.append([i, X.columns[i], v])  
  
importance\_list = sorted(importance\_list, reverse=True, key=lambda x: x[2])  
importance\_list  
  
  
# In[40]:  
  
  
# plotting the bar chart for feature importance  
plt.bar([x for x in range(len(importances))], importances)  
plt.title(**'Feature Importance'**)  
plt.savefig(**'preprocessing\_plots/feature\_importance.png'**, facecolor=**'white'**)  
plt.show()  
  
  
# In[41]:  
  
  
# getting indices of the top 12 important features  
indices = np.argsort(importances)[-1:-13:-1]  
indices  
  
  
# In[42]:  
  
  
# assigning columns of only the top 12 important features to X  
X = X.iloc[:, indices]  
X  
  
  
# In[43]:  
  
  
# importing train\_test\_split from scikit-learn library  
from sklearn.model\_selection import train\_test\_split  
  
# splitting data into random train and test subsets  
# train set - 80%, test set - 20%  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)  
  
  
# In[44]:  
  
  
X\_train  
# shape - (835, 12)  
  
  
# In[45]:  
  
  
X\_test  
# shape - (209, 12)  
  
  
# In[46]:  
  
  
y\_train  
# shape - (835, 1)  
  
  
# In[47]:  
  
  
y\_test  
# shape - (209, 1)  
  
  
# In[48]:  
  
  
# importing KNeighborsClassifier from scikit-learn library  
# KNN is a simple machine learning algorithm based on supervised learning  
# it can be used for both regression and classification problems  
from sklearn.neighbors import KNeighborsClassifier  
  
# finding the best value for k with the minimum error  
  
error = []  
  
for i in range(1, 40):  
 knn = KNeighborsClassifier(n\_neighbors=i)  
 knn.fit(X\_train, y\_train)  
 pred\_i = knn.predict(X\_test)  
 error.append(np.mean(pred\_i != y\_test))  
  
best\_k = error.index(min(error)) + 1  
best\_k  
  
  
# In[49]:  
  
  
# plotting the k values with the error rate  
plt.figure(figsize=(12, 6))  
plt.plot(range(1, 40), error, color=**'red'**, linestyle=**'dashed'**, marker=**'o'**, markerfacecolor=**'blue'**, markersize=10)  
plt.title(**'knn - error rate for k value'**)  
plt.xlabel(**'k value'**)  
plt.ylabel(**'mean error'**)  
plt.savefig(**'preprocessing\_plots/knn\_mean\_error\_for\_k.png'**, facecolor=**'white'**)  
plt.show()  
  
  
# In[50]:  
  
  
# training the KNN model  
knn\_model = KNeighborsClassifier(n\_neighbors=best\_k)  
knn\_model.fit(X\_train, y\_train)  
  
  
# In[51]:  
  
  
# returning parameters of the trained model  
knn\_model.get\_params()  
  
  
# In[52]:  
  
  
# importing pickle module  
# used for serializing and deserializing a python object structure  
import pickle  
  
# saving the trained model to a pickle file  
with open(**'models/knn.pkl'**, **'wb'**) as model\_file:  
 pickle.dump(knn\_model, model\_file)  
  
  
# In[53]:  
  
  
# loading the trained model from the pickle file  
with open(**'models/knn.pkl'**, **'rb'**) as model\_file:  
 knn\_model = pickle.load(model\_file)  
  
  
# In[54]:  
  
  
# predicting labels of X\_test data values on the basis of the trained model  
y\_pred\_knn = knn\_model.predict(X\_test)  
y\_pred\_knn  
  
  
# In[55]:  
  
  
# importing mean\_squared\_error from scikit-learn library  
from sklearn.metrics import mean\_squared\_error  
  
# mean squared error (MSE)  
print(mean\_squared\_error(y\_test, y\_pred\_knn))  
  
# root mean squared error (RMSE)  
# square root of the average of squared differences between predicted and actual value of variable  
print(mean\_squared\_error(y\_test, y\_pred\_knn, squared=False))  
  
  
# In[56]:  
  
  
# importing mean\_absolute\_error from scikit-learn library  
from sklearn.metrics import mean\_absolute\_error  
  
# mean absolute error (MAE)  
mean\_absolute\_error(y\_test, y\_pred\_knn)  
  
  
# In[57]:  
  
  
# importing accuracy\_score from scikit-learn library  
from sklearn.metrics import accuracy\_score  
  
# accuracy  
# ratio of the number of correct predictions to the total number of input samples  
accuracy\_score(y\_test, y\_pred\_knn)  
  
  
# In[58]:  
  
  
# importing precision\_recall\_fscore\_support from scikit-learn library  
from sklearn.metrics import precision\_recall\_fscore\_support  
  
# computing precision, recall, f-measure and support for each class  
print(precision\_recall\_fscore\_support(y\_test, y\_pred\_knn, average=**'micro'**))  
print(precision\_recall\_fscore\_support(y\_test, y\_pred\_knn, average=**'macro'**))  
print(precision\_recall\_fscore\_support(y\_test, y\_pred\_knn, average=**'weighted'**))  
  
  
# In[59]:  
  
  
# importing classification\_report from scikit-learn library  
# used to measure the quality of predictions from a classification algorithm  
from sklearn.metrics import classification\_report  
  
target\_names = list(map(str, sorted(y.unique().tolist())))  
  
# report shows the main classification metrics precision, recall and f1-score on a per-class basis  
print(classification\_report(y\_test, y\_pred\_knn, target\_names=target\_names))  
  
  
# In[60]:  
  
  
# importing confusion\_matrix from scikit-learn library  
from sklearn.metrics import confusion\_matrix  
  
# confusion matrix is a summarized table used to assess the performance of a classification model  
# number of correct and incorrect predictions are summarized with their count according to each class  
print(confusion\_matrix(y\_test, y\_pred\_knn))  
  
  
# In[61]:  
  
  
# importing plot\_confusion\_matrix from scikit-learn library  
from sklearn.metrics import plot\_confusion\_matrix  
  
titles\_options = [(**'knn\_confusion\_matrix\_without\_normalization'**, None),  
 (**'knn\_normalized\_confusion\_matrix'**, **'true'**)]  
  
# plotting the confusion matrix  
for title, normalize in titles\_options:  
 cm\_plot = plot\_confusion\_matrix(knn\_model, X\_test, y\_test, display\_labels=target\_names, cmap=plt.cm.Blues,  
 normalize=normalize)  
 cm\_plot.ax\_.set\_title(title)  
 plt.savefig(**f'preprocessing\_plots/**{title}**.png'**, facecolor=**'white'**)  
  
plt.show()  
  
  
# In[62]:  
  
  
# plotting scatter plot to visualize overlapping of predicted and test target data points  
plt.scatter(range(len(y\_pred\_knn)), y\_pred\_knn, color=**'red'**)  
plt.scatter(range(len(y\_test)), y\_test, color=**'green'**)  
plt.title(**'knn - predicted status vs. real status'**)  
plt.xlabel(**'students'**)  
plt.ylabel(**'status'**)  
plt.savefig(**'preprocessing\_plots/knn\_predicted\_vs\_real.png'**, facecolor=**'white'**)  
plt.show()  
  
  
# In[63]:  
  
  
# importing DecisionTreeClassifier from scikit-learn library  
# decision tree algorithm uses a decision tree as a predictive model to go from observations  
# about an item represented in the branches to conclusions about the item's target value  
# represented in the leaves  
from sklearn.tree import DecisionTreeClassifier  
  
# importing GridSearchCV from scikit-learn library  
# used to loop through predefined hyper-parameters and fit the model on the training set  
from sklearn.model\_selection import GridSearchCV  
  
# importing warnings library to handle exceptions, errors, and warning of the program  
import warnings  
  
# ignoring potential warnings of the program  
warnings.filterwarnings(**'ignore'**)  
  
param\_grid = {  
 **'criterion'**: [**'gini'**, **'entropy'**]  
}  
  
dt\_model = DecisionTreeClassifier()  
  
# looping through criterion hyper-parameters (gini and entropy) of decision tree algorithm  
# and fit the model on the training set  
grid\_search\_cv = GridSearchCV(dt\_model, param\_grid, scoring=**'f1'**, cv=5)  
grid\_search\_cv.fit(X\_train, y\_train)  
  
# printing the best value for the parameters  
grid\_search\_cv.best\_params\_  
  
  
# In[64]:  
  
  
# training the decision tree model  
# default criterion hyper-parameter value is gini  
dt\_model = DecisionTreeClassifier()  
dt\_model.fit(X\_train, y\_train)  
  
  
# In[65]:  
  
  
# returning parameters of the trained model  
dt\_model.get\_params()  
  
  
# In[66]:  
  
  
# getting depth of the decision tree  
print(dt\_model.get\_depth())  
  
# getting number of leaves of the decision tree  
print(dt\_model.get\_n\_leaves())  
  
  
# In[67]:  
  
  
# saving the trained model to a pickle file  
with open(**'models/dt.pkl'**, **'wb'**) as model\_file:  
 pickle.dump(dt\_model, model\_file)  
  
  
# In[68]:  
  
  
# loading the trained model from the pickle file  
with open(**'models/dt.pkl'**, **'rb'**) as model\_file:  
 dt\_model = pickle.load(model\_file)  
  
  
# In[69]:  
  
  
# predicting labels of X\_test data values on the basis of the trained model  
y\_pred\_dt = dt\_model.predict(X\_test)  
y\_pred\_dt  
  
  
# In[70]:  
  
  
# mean squared error (MSE)  
print(mean\_squared\_error(y\_test, y\_pred\_dt))  
  
# root mean squared error (RMSE)  
# square root of the average of squared differences between predicted and actual value of variable  
print(mean\_squared\_error(y\_test, y\_pred\_dt, squared=False))  
  
  
# In[71]:  
  
  
# mean absolute error (MAE)  
mean\_absolute\_error(y\_test, y\_pred\_dt)  
  
  
# In[72]:  
  
  
# accuracy  
# ratio of the number of correct predictions to the total number of input samples  
accuracy\_score(y\_test, y\_pred\_dt)  
  
  
# In[73]:  
  
  
# computing precision, recall, f-measure and support for each class  
print(precision\_recall\_fscore\_support(y\_test, y\_pred\_dt, average=**'micro'**))  
print(precision\_recall\_fscore\_support(y\_test, y\_pred\_dt, average=**'macro'**))  
print(precision\_recall\_fscore\_support(y\_test, y\_pred\_dt, average=**'weighted'**))  
  
  
# In[74]:  
  
  
target\_names = list(map(str, sorted(y.unique().tolist())))  
  
# report shows the main classification metrics precision, recall and f1-score on a per-class basis  
print(classification\_report(y\_test, y\_pred\_dt, target\_names=target\_names))  
  
  
# In[75]:  
  
  
# confusion matrix is a summarized table used to assess the performance of a classification model  
# number of correct and incorrect predictions are summarized with their count according to each class  
print(confusion\_matrix(y\_test, y\_pred\_dt))  
  
  
# In[76]:  
  
  
titles\_options = [(**'dt\_confusion\_matrix\_without\_normalization'**, None),  
 (**'dt\_normalized\_confusion\_matrix'**, **'true'**)]  
  
# plotting the confusion matrix  
for title, normalize in titles\_options:  
 cm\_plot = plot\_confusion\_matrix(dt\_model, X\_test, y\_test, display\_labels=target\_names, cmap=plt.cm.Blues,  
 normalize=normalize)  
 cm\_plot.ax\_.set\_title(title)  
 plt.savefig(**f'preprocessing\_plots/**{title}**.png'**, facecolor=**'white'**)  
  
plt.show()  
  
  
# In[77]:  
  
  
# plotting scatter plot to visualize overlapping of predicted and test target data points  
plt.scatter(range(len(y\_pred\_dt)), y\_pred\_dt, color=**'red'**)  
plt.scatter(range(len(y\_test)), y\_test, color=**'green'**)  
plt.title(**'dt - predicted status vs. real status'**)  
plt.xlabel(**'students'**)  
plt.ylabel(**'status'**)  
plt.savefig(**'preprocessing\_plots/dt\_predicted\_vs\_real.png'**, facecolor=**'white'**)  
plt.show()  
  
  
# In[78]:  
  
  
# importing graphviz library  
# used to create graph objects which can be completed using different nodes and edges  
import graphviz  
  
# importing export\_graphviz from scikit-learn library  
# used to export a decision tree in DOT format to work with the graphviz library  
from sklearn.tree import export\_graphviz  
  
# plotting the decision tree graph  
graphviz\_plot = export\_graphviz(dt\_model, feature\_names=X.columns)  
graph = graphviz.Source(graphviz\_plot)  
graph.render(filename=**'preprocessing\_plots/dt\_graphviz\_plot'**, format=**'png'**, cleanup=True)  
graph.view()  
  
  
# In[79]:  
  
  
# displaying the decision tree graph  
graph  
  
  
# In[80]:  
  
  
# importing RandomForestClassifier from scikit-learn library  
# random forest is a supervised learning algorithm  
# it is used for both classification and regression  
# it is comprised of multiple decision trees  
from sklearn.ensemble import RandomForestClassifier  
  
# n\_estimators - number of trees to be built before taking maximum voting or averages of predictions  
n\_estimators = list(range(1, 150, 3))  
  
param\_grid = {  
 **'n\_estimators'**: n\_estimators  
}  
  
rf\_model = RandomForestClassifier(oob\_score=True)  
  
# looping through n\_estimators hyper-parameters (sequence of numbers from 1 to 149, increment by 3)  
# of random forest algorithm and fit the model on the training set  
grid\_search\_cv = GridSearchCV(rf\_model, param\_grid, cv=5)  
grid\_search\_cv.fit(X, y)  
  
# printing the best value for the parameters  
grid\_search\_cv.best\_params\_  
  
  
# In[81]:  
  
  
# printing the mean\_test\_score of cv\_results\_  
grid\_search\_cv.cv\_results\_[**'mean\_test\_score'**]  
  
  
# In[82]:  
  
  
scores = grid\_search\_cv.cv\_results\_[**'mean\_test\_score'**]  
  
# plotting the accuracy against n\_estimators value for random forest classifier model  
plt.plot(n\_estimators, scores)  
plt.title(**'rf - accuracy and n\_estimators'**)  
plt.xlabel(**'n\_estimators'**)  
plt.ylabel(**'accuracy'**)  
plt.savefig(**'preprocessing\_plots/rf\_accuracy\_n\_estimators.png'**, facecolor=**'white'**)  
plt.show()  
  
  
# In[83]:  
  
  
# training the random forest model  
rf\_model = RandomForestClassifier(n\_estimators=grid\_search\_cv.best\_params\_[**'n\_estimators'**], oob\_score=True)  
rf\_model.fit(X\_train, y\_train)  
  
  
# In[84]:  
  
  
# returning parameters of the trained model  
rf\_model.get\_params()  
  
  
# In[85]:  
  
  
# getting n\_estimators value of the trained model  
rf\_model.n\_estimators  
  
  
# In[86]:  
  
  
# getting oob\_score\_ value of the trained model  
# oob\_score\_ - out-of-bag score  
rf\_model.oob\_score\_  
  
  
# In[87]:  
  
  
# saving the trained model to a pickle file  
with open(**'models/rf.pkl'**, **'wb'**) as model\_file:  
 pickle.dump(rf\_model, model\_file)  
  
  
# In[88]:  
  
  
# loading the trained model from the pickle file  
with open(**'models/rf.pkl'**, **'rb'**) as model\_file:  
 rf\_model = pickle.load(model\_file)  
  
  
# In[89]:  
  
  
# predicting labels of X\_test data values on the basis of the trained model  
y\_pred\_rf = rf\_model.predict(X\_test)  
y\_pred\_rf  
  
  
# In[90]:  
  
  
# mean squared error (MSE)  
print(mean\_squared\_error(y\_test, y\_pred\_rf))  
  
# root mean squared error (RMSE)  
# square root of the average of squared differences between predicted and actual value of variable  
print(mean\_squared\_error(y\_test, y\_pred\_rf, squared=False))  
  
  
# In[91]:  
  
  
# mean absolute error (MAE)  
mean\_absolute\_error(y\_test, y\_pred\_rf)  
  
  
# In[92]:  
  
  
# accuracy  
# ratio of the number of correct predictions to the total number of input samples  
accuracy\_score(y\_test, y\_pred\_rf)  
  
  
# In[93]:  
  
  
# computing precision, recall, f-measure and support for each class  
print(precision\_recall\_fscore\_support(y\_test, y\_pred\_rf, average=**'micro'**))  
print(precision\_recall\_fscore\_support(y\_test, y\_pred\_rf, average=**'macro'**))  
print(precision\_recall\_fscore\_support(y\_test, y\_pred\_rf, average=**'weighted'**))  
  
  
# In[94]:  
  
  
target\_names = list(map(str, sorted(y.unique().tolist())))  
  
# report shows the main classification metrics precision, recall and f1-score on a per-class basis  
print(classification\_report(y\_test, y\_pred\_rf, target\_names=target\_names))  
  
  
# In[95]:  
  
  
# confusion matrix is a summarized table used to assess the performance of a classification model  
# number of correct and incorrect predictions are summarized with their count according to each class  
print(confusion\_matrix(y\_test, y\_pred\_rf))  
  
  
# In[96]:  
  
  
titles\_options = [(**'rf\_confusion\_matrix\_without\_normalization'**, None),  
 (**'rf\_normalized\_confusion\_matrix'**, **'true'**)]  
  
# plotting the confusion matrix  
for title, normalize in titles\_options:  
 cm\_plot = plot\_confusion\_matrix(rf\_model, X\_test, y\_test, display\_labels=target\_names, cmap=plt.cm.Blues,  
 normalize=normalize)  
 cm\_plot.ax\_.set\_title(title)  
 plt.savefig(**f'preprocessing\_plots/**{title}**.png'**, facecolor=**'white'**)  
  
plt.show()  
  
  
# In[97]:  
  
  
# plotting scatter plot to visualize overlapping of predicted and test target data points  
plt.scatter(range(len(y\_pred\_rf)), y\_pred\_rf, color=**'red'**)  
plt.scatter(range(len(y\_test)), y\_test, color=**'green'**)  
plt.title(**'rf - predicted status vs. real status'**)  
plt.xlabel(**'students'**)  
plt.ylabel(**'status'**)  
plt.savefig(**'preprocessing\_plots/rf\_predicted\_vs\_real.png'**, facecolor=**'white'**)  
plt.show()  
  
  
# In[98]:  
  
  
# there are mainly three types of Naive Bayes models  
# they are GaussianNB, MultinomialNB, and BernoulliNB  
# importing GaussianNB from scikit-learn library  
from sklearn.naive\_bayes import GaussianNB  
  
# training the gaussian naive bayes model  
nb\_model = GaussianNB()  
nb\_model.fit(X\_train, y\_train)  
  
  
# In[99]:  
  
  
# returning parameters of the trained model  
nb\_model.get\_params()  
  
  
# In[100]:  
  
  
# saving the trained model to a pickle file  
with open(**'models/nb.pkl'**, **'wb'**) as model\_file:  
 pickle.dump(nb\_model, model\_file)  
  
  
# In[101]:  
  
  
# loading the trained model from the pickle file  
with open(**'models/nb.pkl'**, **'rb'**) as model\_file:  
 nb\_model = pickle.load(model\_file)  
  
  
# In[102]:  
  
  
# predicting labels of X\_test data values on the basis of the trained model  
y\_pred\_nb = nb\_model.predict(X\_test)  
y\_pred\_nb  
  
  
# In[103]:  
  
  
# mean squared error (MSE)  
print(mean\_squared\_error(y\_test, y\_pred\_nb))  
  
# root mean squared error (RMSE)  
# square root of the average of squared differences between predicted and actual value of variable  
print(mean\_squared\_error(y\_test, y\_pred\_nb, squared=False))  
  
  
# In[104]:  
  
  
# mean absolute error (MAE)  
mean\_absolute\_error(y\_test, y\_pred\_nb)  
  
  
# In[105]:  
  
  
# accuracy  
# ratio of the number of correct predictions to the total number of input samples  
accuracy\_score(y\_test, y\_pred\_nb)  
  
  
# In[106]:  
  
  
# computing precision, recall, f-measure and support for each class  
print(precision\_recall\_fscore\_support(y\_test, y\_pred\_nb, average=**'micro'**))  
print(precision\_recall\_fscore\_support(y\_test, y\_pred\_nb, average=**'macro'**))  
print(precision\_recall\_fscore\_support(y\_test, y\_pred\_nb, average=**'weighted'**))  
  
  
# In[107]:  
  
  
target\_names = list(map(str, sorted(y.unique().tolist())))  
  
# report shows the main classification metrics precision, recall and f1-score on a per-class basis  
print(classification\_report(y\_test, y\_pred\_nb, target\_names=target\_names))  
  
  
# In[108]:  
  
  
# confusion matrix is a summarized table used to assess the performance of a classification model  
# number of correct and incorrect predictions are summarized with their count according to each class  
print(confusion\_matrix(y\_test, y\_pred\_nb))  
  
  
# In[109]:  
  
  
titles\_options = [(**'nb\_confusion\_matrix\_without\_normalization'**, None),  
 (**'nb\_normalized\_confusion\_matrix'**, **'true'**)]  
  
# plotting the confusion matrix  
for title, normalize in titles\_options:  
 cm\_plot = plot\_confusion\_matrix(nb\_model, X\_test, y\_test, display\_labels=target\_names, cmap=plt.cm.Blues,  
 normalize=normalize)  
 cm\_plot.ax\_.set\_title(title)  
 plt.savefig(**f'preprocessing\_plots/**{title}**.png'**, facecolor=**'white'**)  
  
plt.show()  
  
  
# In[110]:  
  
  
# plotting scatter plot to visualize overlapping of predicted and test target data points  
plt.scatter(range(len(y\_pred\_nb)), y\_pred\_nb, color=**'red'**)  
plt.scatter(range(len(y\_test)), y\_test, color=**'green'**)  
plt.title(**'nb - predicted status vs. real status'**)  
plt.xlabel(**'students'**)  
plt.ylabel(**'status'**)  
plt.savefig(**'preprocessing\_plots/nb\_predicted\_vs\_real.png'**, facecolor=**'white'**)  
plt.show()  
  
  
# In[111]:  
  
  
# importing recall\_score, precision\_score, f1\_score from scikit-learn library  
from sklearn.metrics import recall\_score, precision\_score, f1\_score  
  
accuracy = {}  
recall = {}  
precision = {}  
f1 = {}  
rmse = {}  
mae = {}  
  
y\_pred\_dict = {  
 **'KNN'**: y\_pred\_knn,  
 **'DT'**: y\_pred\_dt,  
 **'RF'**: y\_pred\_rf,  
 **'NB'**: y\_pred\_nb,  
}  
  
for y\_pred in y\_pred\_dict:  
 accuracy[y\_pred] = accuracy\_score(y\_test, y\_pred\_dict[y\_pred])  
 recall[y\_pred] = recall\_score(y\_test, y\_pred\_dict[y\_pred], average=**'weighted'**)  
 precision[y\_pred] = precision\_score(y\_test, y\_pred\_dict[y\_pred], average=**'weighted'**)  
 f1[y\_pred] = f1\_score(y\_test, y\_pred\_dict[y\_pred], average=**'weighted'**)  
 rmse[y\_pred] = mean\_squared\_error(y\_test, y\_pred\_dict[y\_pred], squared=False)  
 mae[y\_pred] = mean\_absolute\_error(y\_test, y\_pred\_dict[y\_pred])  
  
# ratio of the number of correct predictions to the total number of input samples  
print(accuracy)  
  
# recall is the ratio tp / (tp + fn)  
# tp is the number of true positives  
# fn the number of false negatives  
print(recall)  
  
# precision is the ratio tp / (tp + fp)  
# tp is the number of true positives  
# fp the number of false positives  
print(precision)  
  
# F1 score is also known as balanced F-score or F-measure  
# F1 score can be interpreted as a weighted average of the precision and recall  
# F1 = 2 \* (precision \* recall) / (precision + recall)  
print(f1)  
  
# root mean squared error (RMSE)  
# square root of the average of squared differences between predicted and actual value of variable  
print(rmse)  
  
# mean absolute error (MAE)  
print(mae)  
  
  
# In[112]:  
  
  
# sorting the accuracy scores of four models in descending order  
sorted(accuracy.items(), key=lambda kv: kv[1], reverse=True)  
  
  
# In[113]:  
  
  
# plotting the accuracy comparison bar chart  
plt.bar(y\_pred\_dict.keys(), accuracy.values(), color=**'rgbc'**)  
plt.title(**'Accuracy Comparison'**)  
plt.xlabel(**'Algorithm'**)  
plt.ylabel(**'Accuracy'**)  
plt.savefig(**'preprocessing\_plots/accuracy\_comparison.png'**, facecolor=**'white'**)  
plt.show()  
  
  
# In[114]:  
  
  
def bar\_plot(ax, data, colors=None, total\_width=0.8, single\_width=1, legend=True):  
 if colors is None:  
 colors = plt.rcParams[**'axes.prop\_cycle'**].by\_key()[**'color'**]  
 n\_bars = len(data)  
 bar\_width = total\_width / n\_bars  
 bars = []  
 for i, (name, values) in enumerate(data.items()):  
 x\_offset = (i - n\_bars / 2) \* bar\_width + bar\_width / 2  
 for x, y in enumerate(values):  
 bar = ax.bar(x + x\_offset, y, width=bar\_width \* single\_width, color=colors[i % len(colors)])  
 bars.append(bar[0])  
 if legend:  
 ax.legend(bars, data.keys())  
  
  
# In[115]:  
  
  
data = {}  
  
for key in y\_pred\_dict.keys():  
 data[key] = [accuracy[key], recall[key], precision[key], f1[key], rmse[key], mae[key]]  
  
fig, ax = plt.subplots()  
bar\_plot(ax, data, total\_width=.9, single\_width=.9)  
  
# plotting the algorithm comparison chart for all evaluation metrics  
plt.title(**'Algorithm Comparison'**)  
plt.xlabel(**'Metrics'**)  
plt.ylabel(**'Scores'**)  
plt.xticks(range(len(data[key])), [**'accuracy'**, **'recall'**, **'precision'**, **'f1'**, **'rmse'**, **'mae'**])  
plt.savefig(**'preprocessing\_plots/algorithm\_comparison.png'**, facecolor=**'white'**)  
plt.show()  
  
  
# In[115]: