Mini Project Report

Entitled

Student dropout prediction

Submitted to the Department of Electronics Engineering in Partial Fulfilment for the Requirements for the Degree of

Bachelor of Technology
(Electronics and Communication)

: Presented & Submitted By :

Suhani Parmar, Srushti Modh

Roll No. (U20EC112, U20EC138)

B. TECH. VI (EC), 6^{th} Semester

: Guided By :

Dr. Kishor Upla

Assistant Professor, SVNIT

(Year: 2022-23)

DEPARTMENT OF ELECTRONICS ENGINEERING

SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY

Surat-395007, Gujarat, INDIA.

Sardar Vallabhbhai National Institute Of Technology

Surat - 395 007, Gujarat, India

DEPARTMENT OF ELECTRONICS ENGINEERING

CERTIFICATE

This is to certify that the Mini-Project Report entitled "**Student dropout prediction**" is presented & submitted by Suhani Parmar, Srushti Modh, bearing Roll No. U20EC112, U20EC138, of B.Tech. VI, 6^{th} Semester in the partial fulfillment of the requirement for the award of B.Tech. Degree in Electronics & Communication Engineering for academic year 2022-23.

They have successfully and satisfactorily completed their **Mini-Project** in all respects. We, certify that the work is comprehensive, complete and fit for evaluation.

Dr. Kishor Upla
Assistant Professor & Project Guide

Abstract

Student dropout prediction refers to the use of data analysis and machine learning techniques to identify students who are at risk of dropping out of school. The prediction of student dropout is a critical task for educational institutions, as it can help them to take proactive measures to prevent students from leaving school prematurely.

The process of student dropout prediction typically involves the collection of various student data, such as demographics, academic performance, and socio-economic factors. These data are then analyzed using predictive models to identify patterns and trends that may indicate students who are at risk of dropping out.

The development of effective dropout prediction models can benefit both students and educational institutions. For students, early identification of those who are at risk of dropping out can lead to targeted interventions and support, such as tutoring, counseling, and mentorship. For educational institutions, identifying students who are at risk of dropping out can help them to allocate resources effectively and implement interventions to improve student retention rates.

In conclusion, student dropout prediction is an essential task for educational institutions to ensure that students receive the support they need to succeed academically and achieve their long-term goals.

Table of Contents

List of Figures

Chapter 1
Introduction

Student dropout is a significant problem that affects the educational system worldwide. Dropout rates refer to the percentage of students who leave school before completing their studies. Students may drop out of school for a variety of reasons, including financial difficulties, family responsibilities, poor academic performance, bullying, or lack of interest in the subject matter.

1.1 Background

Dropout students are more likely to experience poverty, unemployment, and poor health outcomes throughout their lives. Additionally, dropouts are at a higher risk of engaging in criminal activities and substance abuse, leading to a negative impact on society as a whole. To tackle the dropout problem, educational institutions and policymakers need to implement effective strategies and support systems for at-risk students. These can include early intervention programs, mentorship and counseling services, academic tutoring, and career guidance. By addressing the underlying causes of student dropout, we can increase the likelihood of students staying in school, completing their education, and achieving their full potential. Several governments have developed and implemented dropout early warning systems to deal with this issue. To list a few measures, the department of education and early childhood development in the state of Victoria in Australia developed the student mapping tool to help private and public schools to analyze students at the risk of disengagement and dropout. Similarly, the state of Wisconsin in the United States developed the same system to predict students' dropouts. Furthermore, dropout early warning systems were implemented in the United States during 2014–2015 for more than half of public high schools (Sullivan, 2017).

Machine learning is a promising tool for building a predictive model for student dropout and offers early warning to responsible authorities to take alternative measures to students at the risk of dropping out the school. Recently, several studies emphasized the prediction of students' performances using data mining (DM)/machine learning (ML) models. For example, student dropout in higher education was predicted using the largest known dataset on higher education attrition, which tracks over 32,500 students' demographics and transcript records at one of the nation's largest public universities and the overall results demonstrated that ML models have a significant impact on

student retention and success while pointing to several promising directions for future work. Similarly, DM methods were employed to analyze student dropouts in the junior years. Among the DM methods used such as logistic regression, decision trees, and neural networks, the decision trees demonstrated the greater ability to predict accurately student dropout (Jadri´c et al., 2010).

1.2 Objective

Our goal for this project is to identify students who might need early intervention before they fail to graduate. And to achieve it, we analyse what factors will affect dropout decisions and create a machine learning model to predict the dropout status of undergraduate students. We also go through various methodologies to compare the different the dropout prediction systems.

Chapter 2
Methodology

We analyze the relationship among students' personal information, academic records and status from previous semesters by using classification techniques. Then we create a model to predict their academic status (dropout or not dropout) in the upcoming semester

2.0.1 Dataset

The dataset used in this project is included as student-data.csv. This dataset has the following attributes:

- school ? student's school (binary: "GP" or "MS")
- sex ? student's sex (binary: "F" - female or "M" - male)
- age ? student's age (numeric: from 15 to 22)
- address ? student's home address type (binary: "U" - urban or "R" - rural)
- famsize ? family size (binary: "LE3" - less or equal to 3 or "GT3" - greater than 3)
- Pstatus ? parent's cohabitation status (binary: "T" - living together or "A" - apart)
- Medu ? mother's education (numeric: 0 - none, 1 - primary education (4th grade), 2 -€“ 5th to 9th grade, 3 - secondary education or 4 -€“ higher education)
- Fedu ? father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 - 5th to 9th grade, 3 - secondary education or 4 -€“ higher education)
- Mjob ? mother's job (nominal: "teacher", "health" care related, civil "services" (e.g. administrative or police), "athome" or "other")
- Fjob ? father's job (nominal: "teacher", "health" care related, civil "services" (e.g. administrative or police), "athome" or "other")
- reason ? reason to choose this school (nominal: close to "home", school "reputation", "course" preference or "other")

- guardian ? student's guardian (nominal: "mother", "father" or "other")
- traveltime ? home to school travel time (numeric: 1 - ¡15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - ¿1 hour)
- studytime ? weekly study time (numeric: 1 - ¡2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - ¿10 hours)
- failures ? number of past class failures (numeric: n if 1¡=n¡3, else 4)
- schoolsup ? extra educational support (binary: yes or no)
- famsup ? family educational support (binary: yes or no)
- paid ? extra paid classes within the course subject (Math or Portuguese) (binary: yes or no)
- activities ? extra-curricular activities (binary: yes or no)
- nursery ? attended nursery school (binary: yes or no)
- higher ? wants to take higher education (binary: yes or no)
- internet ? Internet access at home (binary: yes or no)
- romantic ? with a romantic relationship (binary: yes or no)
- famrel ? quality of family relationships (numeric: from 1 - very bad to 5 - excellent)
- freetime ? free time after school (numeric: from 1 - very low to 5 - very high)
- goout ? going out with friends (numeric: from 1 - very low to 5 - very high)
- Dalc ? workday alcohol consumption (numeric: from 1 - very low to 5 - very high)
- Walc ? weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)
- health ? current health status (numeric: from 1 - very bad to 5 - very good)
- absences ? number of school absences (numeric: from 0 to 93)
- passed ? did the student pass the final exam (binary: yes or no)

ıool	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	internet	romantic	famrel	freetime	goout	Dalc	Walc	health	absences	passed
GP	F	18	U	GT3	A	4	4	at_home	teacher	...	no	no	4	3	4	1	1	3	6	no
GP	F	17	U	GT3	T	1	1	at_home	other	...	yes	no	5	3	3	1	1	3	4	no
GP	F	15	U	LE3	T	1	1	at_home	other	...	yes	no	4	3	2	2	3	3	10	yes
GP	F	15	U	GT3	T	4	2	health	services	...	yes	yes	3	2	2	1	1	5	2	yes
GP	F	16	U	GT3	T	3	3	other	other	...	no	no	4	3	2	1	2	5	4	yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
MS	M	20	U	LE3	A	2	2	services	services	...	no	no	5	5	4	4	5	4	11	no
MS	M	17	U	LE3	T	3	1	services	services	...	yes	no	2	4	5	3	4	2	3	yes
MS	M	21	R	GT3	T	1	1	other	other	...	no	no	5	5	3	3	3	3	3	no
MS	M	18	R	LE3	T	3	2	services	other	...	yes	no	4	4	1	3	4	5	0	yes
MS	M	19	U	LE3	T	1	1	other	at_home	...	yes	no	3	2	3	3	3	5	5	no

: × 31 columns

Figure 2.1: dataset

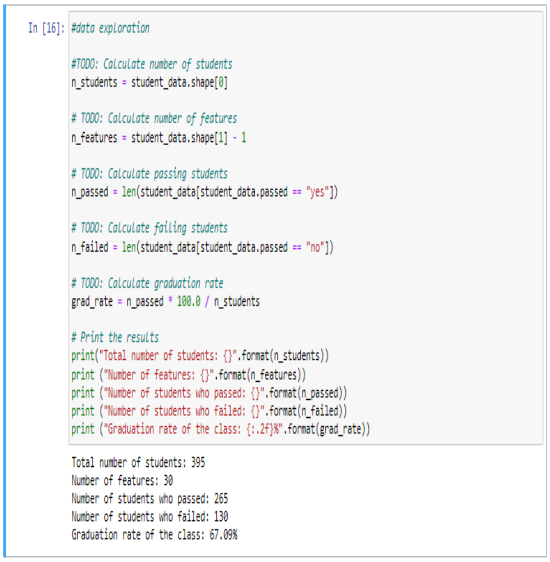
2.0.2 Exploring data

we run the code cell below to load necessary Python libraries and load the student data. Note that the last column from this dataset, 'passed', will be our target label (whether the student graduated or didn't graduate). All other columns are features about each student.

In [15]:
```
import numpy as np
import pandas as pd
from time import time
from sklearn.metrics import f1_score


student_data=pd.read_csv(r"C:\Users\SUHANI\OneDrive\Desktop\studentdata.csv")
student_data
```

Figure 2.2: importing libraries



```
In [16]: #data exploration

         #TODO: Calculate number of students
         n_students = student_data.shape[0]

         # TODO: Calculate number of features
         n_features = student_data.shape[1] - 1

         # TODO: Calculate passing students
         n_passed = len(student_data[student_data.passed == "yes"])

         # TODO: Calculate failing students
         n_failed = len(student_data[student_data.passed == "no"])

         # TODO: Calculate graduation rate
         grad_rate = n_passed * 100.0 / n_students

         # Print the results
         print("Total number of students: {}".format(n_students))
         print ("Number of features: {}".format(n_features))
         print ("Number of students who passed: {}".format(n_passed))
         print ("Number of students who failed: {}".format(n_failed))
         print ("Graduation rate of the class: {:.2f}%".format(grad_rate))

         Total number of students: 395
         Number of features: 30
         Number of students who passed: 265
         Number of students who failed: 130
         Graduation rate of the class: 67.09%
```

Figure 2.3: data exploration

2.0.3 Data preparation

It is often the case that the data you obtain contains non-numeric features. This can be a problem, as most machine learning algorithms expect numeric data to perform computations with.

We separate out student info feature and target columns to see if any other feature is non-numeric

2.0.4 Preprocessing

there are several non-numeric columns that need to be converted! Many of them are simply yes/no, e.g. internet. These can be reasonably converted into 1/0 (binary) values.

Other columns, like Mjob and Fjob, have more than two values, and are known as categorical variables. The recommended way to handle such a column is to create as many columns as possible values (e.g. Fjobteacher, Fjobother, Fjobservices, etc.), and assign a 1 to one of them and 0 to all others. These generated columns are sometimes called dummy variables, and we will use the pandas.gedummies() function to perform this transformation.

In [18]:
```
#data preparation

# Extract feature columns
feature_cols = list(student_data.columns[:-1])

# Extract target column 'passed'
target_col = student_data.columns[-1]

# Show the list of columns
print ("Feature columns:\n{}".format(feature_cols))
print ("\nTarget column: {}".format(target_col))

# Separate the data into feature data and target data (X_all and y_all, respectively)
X_all = student_data[feature_cols]
y_all = student_data[target_col]

# Show the feature information by printing the first five rows
print ("Feature values:")
print (X_all.head())
```

```
Feature columns:
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob', 'Fjob', 'reason',
'guardian', 'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nurs
ery', 'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'abs
ences']

Target column: passed
Feature values:
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  ...  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  ...
1     GP   F   17       U     GT3       T     1     1  at_home     other  ...
2     GP   F   15       U     LE3       T     1     1  at_home     other  ...
3     GP   F   15       U     GT3       T     4     2   health  services  ...
4     GP   F   16       U     GT3       T     3     3    other     other  ...

  higher internet  romantic  famrel  freetime goout Dalc Walc health absences
0    yes       no        no       4         3     4    1    1      3        6
1    yes      yes        no       5         3     3    1    1      3        4
2    yes      yes        no       4         3     2    2    3      3       10
3    yes      yes       yes       3         2     2    1    1      5        2
4    yes       no        no       4         3     2    1    2      5        4

[5 rows x 30 columns]
```

Figure 2.4: data preparation

In [19]:
```
def preprocess_features(X):
    ''' Preprocesses the student data and converts non-numeric binary variables into
        binary (0/1) variables. Converts categorical variables into dummy variables. '''

    # Initialize new output DataFrame
    output = pd.DataFrame(index = X.index)

    # Investigate each feature column for the data
    for col, col_data in X.iteritems():

        # If data type is non-numeric, replace all yes/no values with 1/0
        if col_data.dtype == object:
            col_data = col_data.replace(['yes', 'no'], [1, 0])

        # If data type is categorical, convert to dummy variables
        if col_data.dtype == object:
            # Example: 'school' => 'school_GP' and 'school_MS'
            col_data = pd.get_dummies(col_data, prefix = col)

        # Collect the revised columns
        output = output.join(col_data)

    return output

X_all = preprocess_features(X_all)
print ("Processed feature columns ({} total features):\n{}".format(len(X_all.columns), list(X_all.columns)))
```

```
Processed feature columns (48 total features):
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'address_U', 'famsize_GT3', 'famsize_LE3', 'Pstatus_A', 'Pstat
us_T', 'Medu', 'Fedu', 'Mjob_at_home', 'Mjob_health', 'Mjob_other', 'Mjob_services', 'Mjob_teacher', 'Fjob_at_home', 'Fjob_heal
th', 'Fjob_other', 'Fjob_services', 'Fjob_teacher', 'reason_course', 'reason_home', 'reason_other', 'reason_reputation', 'guard
ian_father', 'guardian_mother', 'guardian_other', 'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid', 'activi
ties', 'nursery', 'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences']
```

Figure 2.5: Preprocessing features

2.0.5 Training and testing data splitting

we have converted all categorical features into numeric values. For the next step, we split the data (both features and corresponding labels) into training and test sets

```
In [21]: # TODO: Import any additional functionality you may need here
         from sklearn.model_selection import train_test_split

         # TODO: Set the number of training points
         num_train = 300

         # Set the number of testing points
         num_test = X_all.shape[0] - num_train

         # TODO: Shuffle and split the dataset into the number of training and testing points above
         X_train, X_test, y_train, y_test = train_test_split(
                X_all, y_all, stratify=y_all, test_size=95, random_state=0)

         # Show the results of the split
         print ("Training set has {} samples.".format(X_train.shape[0]))
         print ("Testing set has {} samples.".format(X_test.shape[0]))

         Training set has 300 samples.
         Testing set has 95 samples.
```

Figure 2.6: Data splitting

2.0.6 Model application

We can use different supervised learning models for this problem. We will go through them one by one:

Decision Trees

Strengths:

- Easy to use
- Graphically intuitive
- can be used for both regression and classification
- fast computing of prediction results

Weaknesses:

- Prone to overfitting
- Unstable: If you change the data a little bit you can possibly end up with a total different tree (using them within an ensemble method can solve this problem)
- No guarantee that we have the global optimal decision tree as it uses greedy algorithms inside (e.g. maximizing information gain)

We don't have a lot of features so it is not super likely that our decision tree will completely overfit. There is also no single attribute value missing which makes it easy for us to just use decision trees. We don't need any preprocsessing. Also we have an offline problem where the data is not changing so we don't have the problem of unstable decision trees as we don't have data changes.

Support Vector Machines

Strength:

- Effective in high dimensional spaces
- performs really good where we have a clear decision boundary
- memory effecient as it only uses support vectors for decision and not all training points

Weaknesses:

- works badly with noisy data
- huge computing time on large datasets

We don't have a big data set with our 395 students and 30 features which means that SVMs can calculate that in a reasonable amount of time. We also don't have any noise in our data set which also favors SVMs.

Naive Bayes Classifier

Strengths:

- extremely fast
- only need a small amount of training data

Weaknesses:

- bad estimator

We don't have that much training data and we want to have a fast (minimum computing time) algorithm. Naive Bayes seems to promise to work with small data and seems to be quite fast.

2.0.7 Setup

We initialize three helper functions which can be used for training and testing three models as discussed above.

```
In [24]: #setup

         def train_classifier(clf, X_train, y_train):
             ''' Fits a classifier to the training data. '''

             # Start the clock, train the classifier, then stop the clock
             start = time()
             clf.fit(X_train, y_train)
             end = time()

             # Print the results
             print ("Trained model in {:.4f} seconds".format(end - start))


         def predict_labels(clf, features, target):
             ''' Makes predictions using a fit classifier based on F1 score. '''

             # Start the clock, make predictions, then stop the clock
             start = time()
             y_pred = clf.predict(features)
             end = time()

             # Print and return results
             print ("Made predictions in {:.4f} seconds.".format(end - start))
             return f1_score(target.values, y_pred, pos_label='yes')


         def train_predict(clf, X_train, y_train, X_test, y_test):
             ''' Train and predict using a classifer based on F1 score. '''

             # Indicate the classifier and the training set size
             print ("Training a {} using a training set size of {}. . .".format(clf.__class__.__name__, len(X_train)))

             # Train the classifier
             train_classifier(clf, X_train, y_train)

             # Print the results of prediction for both training and testing
             print ("F1 score for training set: {:.4f}.".format(predict_labels(clf, X_train, y_train)))
             print ("F1 score for test set: {:.4f}.".format(predict_labels(clf, X_test, y_test)))
```

Figure 2.7: model setup

2.0.8 Implementation

We need to train and predict on each classifier for three different data set sizes:100,200 and 300. Hence we expect to get 9 outputs, 3 per model.

In [25]:
```
#model implementation

from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.naive_bayes import GaussianNB

# TODO: Initialize the three models
clf_A = DecisionTreeClassifier(random_state=0)
clf_B = svm.SVC(random_state=0)
clf_C = GaussianNB()

# TODO: Set up the training set sizes
X_train_100 = X_train[0:100]
y_train_100 = y_train[0:100]

X_train_200 = X_train[0:200]
y_train_200 = y_train[0:200]

X_train_300 = X_train[0:300]
y_train_300 = y_train[0:300]

# TODO: Execute the 'train_predict' function for each classifier and each training set size
# train_predict(clf, X_train, y_train, X_test, y_test)

classifiers = [clf_A, clf_B, clf_C]
training_subsets = [(X_train_100, y_train_100), (X_train_200, y_train_200),
                    (X_train_300, y_train_300)]
for c in classifiers:
    for X_train_subset, y_train_subset in training_subsets:
        print (train_predict(c, X_train_subset, y_train_subset, X_test, y_test))
```

Figure 2.8: implementation

** Classifer 1 - Decision Tree**

Training Set Size	Prediction Time (train)	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0003	0.0004	1	0.6942
200	0.0002	0.0002	1	0.7132
300	0.0003	0.0004	1	0.7167

** Classifer 2 - SVM**

Training Set Size	Prediction Time (train)	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0010	0.0007	0.8591	0.7838
200	0.0023	0.0013	0.8693	0.7755
300	0.0061	0.0020	0.8692	0.7586

** Classifer 3 - Gaussian Naive Bayes**

Training Set Size	Prediction Time (train)	Prediction Time (test)	F1 Score (train)	F1 Score (test)
100	0.0005	0.0003	0.8550	0.7481
200	0.0005	0.0003	0.8321	0.7132
300	0.0003	0.0002	0.8088	0.7500

Figure 2.9: Tabular results

2.1 Choosing the best model

We chose a decision tree model because it delivers us a good result without any tuning weather a student will drop out or not, while still being fast in computing time which means we save money on renting servers. It is also very easy to use and leads to higher prediction accuracy

A decision tree is build from top to bottom and consist of nodes. A node can have other nodes as children. If a node does not have any children we will call it leaf. Like in nature at the top is a root node where everything starts.

Constructing a decision tree works like the following:

As input we take the training data with its different features.

We now try to split our data in homogenous chunks. For achieving that we look at each of our attributes how high is the information gain if we would split on this attribute. Information gain in laymans terms means how much I know more when I ask this question. If you compare it to the game where you have to guess who you are general questions like ”Am I alive / Am I a human / ...” will give you a high information gain as it eliminates A LOT of possibilities who you could be. Asking more specific questions as ”Am I Michael Jackson/ Am I born in Berlin” will give you a smaller information gain in the beginning.

We choose the attribute with the highest information gain and split on that attribute which means we create a node which represents our attribute question (e.g. ”male?”). This node is connected to the chunks where we divided our data. We then look at our chunkgs: If we have an entropy of 0 (no more information gain) in our chunk this will be a leaf. This leaf will also save the information to which class our test example belongs. If the entropy is higher then we have to split this chunk further in other chunks . To do this we just apply the same method from 2. and 3. with our chunk as data input.

Predicting works like playing a game where you have to answer questions. The questions are all encoded in the nodes. A question could be ”Age ¿ 42” or ”male?”. Your answer to that question leads you to a different node where either another questions waits for you or if there is no question anymore then you are in a leaf. The leaf is attached with a class which marks also the prediction class of your input.

2.2 Model tuning

we take the entire training set ftor this and implement the following:

```
#model tuning

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, f1_score
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier

# NOTE: As I could not tune any parameters for the GaussianNB I decided to do the parameter
# optimization for my second choice the decision tree classifier.


# TODO: Create the parameters list you wish to tune
parameters = {}
parameters_dt = {'min_samples_split':(2,3,4,5,6,7,8,9,10, 11, 12),
                 'max_depth' :(1,2,3,4,5,6,7,8,9,10),
                 'splitter': ('best', 'random')}

# TODO: Initialize the classifier
clf = GaussianNB()
clf_dt = DecisionTreeClassifier()

# TODO: Make an f1 scoring function using 'make_scorer'
f1_scorer = make_scorer(f1_score, pos_label='yes')

# TODO: Perform grid search on the classifier using the f1_scorer as the scoring method
grid_obj = GridSearchCV(clf, parameters, scoring=f1_scorer)
grid_obj_dt = GridSearchCV(clf_dt, parameters_dt, scoring=f1_scorer)

# TODO: Fit the grid search object to the training data and find the optimal parameters
grid_obj = grid_obj.fit(X_train, y_train)
grid_obj_dt = grid_obj_dt.fit(X_train, y_train)


print (grid_obj_dt.best_params_)

# Get the estimator
clf = grid_obj.best_estimator_
clf_dt = grid_obj_dt.best_estimator_

# Report the final F1 score for training and testing after parameter tuning
print (("Tuned model has a training F1 score of {:.4f}.".format(predict_labels(clf, X_train, y_train))))
print (("Tuned model has a testing F1 score of {:.4f}.".format(predict_labels(clf, X_test, y_test))))

print ("Tuned decision tree model has a training F1 score of {:.4f}.".format(predict_labels(clf_dt, X_train, y_train)))
print ("Tuned decision tree model has a testing F1 score of {:.4f}.".format(predict_labels(clf_dt, X_test, y_test)))
```

Figure 2.10: model tuning

```
{'max_depth': 1, 'min_samples_split': 5, 'splitter': 'random'}
Made predictions in 0.0037 seconds.
Tuned model has a training F1 score of 0.8000.
Made predictions in 0.0064 seconds.
Tuned model has a testing F1 score of 0.7273.
Made predictions in 0.0000 seconds.
Tuned decision tree model has a training F1 score of 0.8270.
Made predictions in 0.0010 seconds.
Tuned decision tree model has a testing F1 score of 0.8105.
```

Figure 2.11: tuning results

Chapter 3
Results

- GaussianNB

Final f1-score for testing is 0.7273 and 0.8 for training. The untuned score with 300 data points was 0.75 for testing which was actually better. The testing error before was 0.8088 which was also slightly better which is kind of suprising. But we acutally did not tune any parameters as the GaussianNB does not have any parameters.

- Decision Tree

Final f1-score for the decision tree is 0.8129 for the testing set which is a lot better than the 0.7167 f1-score of our untuned decison tree model. The training score is now 0.8098 compared to a 1.0 score on our untuned model. You can clearly see how we went from a overfitting model to a better generalizing model just by tuning the parameters.

Conclusion

The effectiveness of a student dropout prediction model would depend on various factors such as the accuracy of the data used to train the model, the quality of the features used to make predictions, and the ability to intervene in time to prevent dropouts.

It's important to note that while a prediction model can provide valuable insights, it should not be used in isolation as the sole basis for decision-making. Educators and administrators should use the results of the model in conjunction with their professional judgment and other relevant information to make informed decisions about how to support at-risk students.

Overall, a well-designed and well-implemented student dropout prediction model has the potential to positively impact student outcomes and reduce dropout rates.

References